Learning the Boundary of Inductive Invariants

YOTAM M. Y. FELDMAN, Tel Aviv University, Israel
MOOLY SAGIV, Tel Aviv University, Israel
SHARON SHOHAM, Tel Aviv University, Israel
JAMES R. WILCOX, Certora, USA

We study the *complexity of invariant inference* and its connections to *exact concept learning*. We define a condition on invariants and their geometry, called the *fence* condition, which permits applying theoretical results from exact concept learning to answer open problems in invariant inference theory. The condition requires the invariant's *boundary*—the states whose Hamming distance from the invariant is one—to be backwards reachable from the bad states in a small number of steps. Using this condition, we obtain the first polynomial complexity result for an interpolation-based invariant inference algorithm, efficiently inferring monotone DNF invariants with access to a SAT solver as an oracle. We further harness Bshouty's seminal result in concept learning to efficiently infer invariants of a larger syntactic class of invariants beyond monotone DNF. Lastly, we consider the robustness of inference under program transformations. We show that some simple transformations preserve the fence condition, and that it is sensitive to more complex transformations.

CCS Concepts: • **Theory of computation → Theory and algorithms for application domains**; **Program verification**; • **Software and its engineering → Formal methods**.

Additional Key Words and Phrases: invariant inference, complexity, exact learning, interpolation, Hamming geometry

1 INTRODUCTION

This paper addresses the problem of inferring inductive invariants to verify the safety of systems, which lies at the foundation of software and hardware verification. This problem has high complexity, even in finite-state systems [Lahiri and Qadeer 2009]. We are particularly interested in SAT-based inductive invariant inference, which harnesses the SAT solver in order to check if candidate invariant is inductive and to perform bounded model checking. Many techniques tackle this challenge [e.g. Alur et al. 2015; Bradley 2011; Dillig et al. 2013; Eén et al. 2011; Fedyukovich and Bodík 2018; McMillan 2003; Srivastava et al. 2013]. Despite their practical achievements, the theoretical understanding of the complexity of these algorithms is lacking.

This work harnesses the theory of exact concept learning to broaden our understanding of the complexity of invariant inference. We focus on the fundamental setting of propositional systems,

Authors' addresses: Yotam M. Y. Feldman, Tel Aviv University, Israel, yotam.feldman@gmail.com; Mooly Sagiv, Tel Aviv University, Israel, msagiv@acm.org; Sharon Shoham, Tel Aviv University, Israel, sharon.shoham@gmail.com; James R. Wilcox, Certora, USA, james@certora.com.
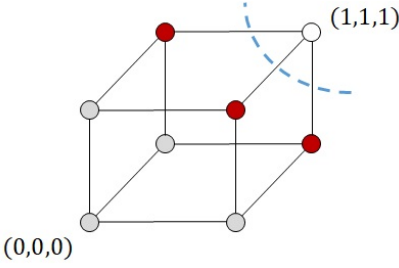
15

Fig. 1. The (outer) boundary of an invariant $I = x \wedge y \wedge z$, denoting the singleton set containing the far-top-right vertex of the 3-dimensional Boolean hypercube, $\{(1, 1, 1)\}$. Its neighbors are I's boundary (depicted in red): $\{(1, 1, 0), (1, 0, 1), (0, 1, 1)\}$. The rest of the vertices are in $\neg I$ but not in the boundary (depicted in gray). (Illustration inspired by [O'Donnell 2014, Fig. 2.1].)
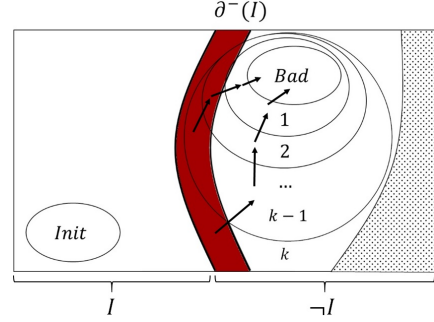

Fig. 2. An illustration of the fence condition. The boundary $\partial^-(I)$ of the invariant (the states in $\neg I$ nearest to I, in red) are backwards k-reachable (reach a bad state in k steps, for example by the transitions depicted by the arrows), but not all states in $\neg I$ are backwards k-reachable (or even backwards reachable at all, in the dotted area).

which is useful also for infinite-state systems through predicate abstraction [Flanagan and Qadeer 2002; Graf and Saïdi 1997].

Our results shed light on the cases in which interpolation-based invariant inference [e.g. McMillan 2003, 2006] behaves well. We analyze an interpolation-based algorithm in which interpolants are computed in a model-based fashion [Bjørner et al. 2013; Chockler et al. 2012], which is inspired by IC3/PDR [Bradley 2011; Eén et al. 2011], and prove an upper bound on the number of iterations until convergence to an invariant. This upper bound is formulated using a condition, which we call the *fence* condition, on the relationship between reachability in the transition system and the geometric notion of the *boundary* of the invariant. Recall that invariants denote sets of states. The boundary of the invariant I is the set of states whose Hamming distance from I is one—states in $\neg I$ where flipping just a single bit results in a state that belongs to I. (See Fig. 1 for an illustration.) The k-fence condition requires that all the states in I's boundary reach a bad state in at most k steps. (See Fig. 2.) Intuitively, the states in the boundary are important, because "perturbing" a single bit is the difference between being included in I and being excluded by it (somewhat analogously to error-correcting codes). Through their backwards reachability in k steps, the fence condition guarantees that the interpolation algorithm, when it uses k for the model checking bound, never "overshoots" to invariants that are weaker than I (even though I itself is unknown to the algorithm). Every safe propositional system admits an invariant that is k-fenced for some finite k, but this is most useful when (1) k is not prohibitively large, and (2) the invariant has a short representation.

The fence condition opens up the possibility to obtain efficiency results for invariant inference. We prove that if the fence condition holds and the invariant has a short disjunctive normal form (DNF) representation that is monotone, i.e., all variables appear only positively, then the interpolation algorithm converges in a linear number of iterations.

We are able to transcend this result using a more complex algorithm, in a sense combining multiple instances of the interpolation-based algorithm, based on work in machine learning by Bshouty [1995]. We prove that when the fence condition holds the algorithm achieves efficient inference for the wider class of *almost-monotone* DNF invariants (DNF invariants that have a constant number of terms that include negative literals).

These theorems are rooted in results on efficient *exact concept learning* of monotone and almost-monotone DNF formulas by Angluin [1987] and Bshouty [1995], respectively. In exact concept

```
init  x₁ = … = xₙ = true                          hot_potato():
                                                      let  xᵢ ≠ xⱼ ∈ J
repeat:                                               if  xᵢ = false ∧ xⱼ = true:
      | hot_potato()                                     xᵢ := true
      | turn_two_on()                                    xⱼ := false
      | havoc_others()

      assert  ¬(x₁ = false, x₂ = … = xₙ = true)   turn_two_on():
                                                      let  xᵢ ≠ xⱼ ∈ J
havoc_others():                                       if  xᵢ = false ∧ xⱼ = false:
      forall  xᵢ ∉ J:                                    xᵢ := true
          xᵢ := *                                        xⱼ := true
```

Fig. 3. An example propositional transition system for which we would like to infer an inductive invariant. The state is over $x_1, \ldots, x_n$. $J \subseteq \{x_1, \ldots, x_n\}$, and we assume $x_1 \in J$. In each iteration, one of the actions (after repeat) is invoked, chosen nondeterministically. `forall` is executed in a single step of the system. `x := *` means that `x` is updated to either true or false nondeterministically.

learning, a learner poses queries to a teacher in order to identify an unknown formula. Unfortunately, invariant inference is harder in general, because the classical queries cannot be implemented [Feldman et al. 2020; Garg et al. 2014]. We show that the fence condition can alleviate these problems when the exact learning algorithm is guaranteed to query only on examples that are positive or have Hamming distance one from a positive example. In fact, we show a general transformation from algorithms for exact concept learning into invariant inference algorithms that obtain analogous correctness results when the fence condition holds. Such transformations are surprising, because in invariant inference, unlike classical concept learning, the teacher—the SAT solver in our case—does not know the target "concept" I, and yet it is still able to "teach" I to the algorithm, thanks to the fence condition. A similar transformation, applied to the CDNF algorithm by Bshouty [1995], provides an algorithm that efficiently infers any invariant that has a short representation in both conjunctive- and disjunctive-normal form (including all small Boolean decision trees) if it satisfies a stronger, two-sided version of the fence condition. Overall, the fence condition bridges the study of efficient exact concept learning and the complexity analysis of invariant inference.

One of the most significant open problems in formal verification is the *robustness* of the inference method with respect to revisions in the code. For example, suppose the inference algorithm succeeded in inferring an invariant, will this hold after a simple change? This problem is very difficult in general; we address it in the context of the interpolation algorithm, and study whether the fence condition holds after a transformation of the program. We show that very simple transformations preserve the fence condition: variable renaming, and taking conjunctions and disjunctions of properties. In contrast, after a new variable is introduced to capture some meaning over the existing variables (in the hope of aiding inference), an invariant that does not refer to the new variable cannot satisfy the fence condition. Similarly, a natural transformation that attempts to reduce non-monotone invariants to monotone ones is unsuccessful.

Our results provide an important step towards understanding the complexity of invariant inference algorithms. It is known that restricting invariants to conjunctions enables efficient inference [Flanagan and Leino 2001; Lahiri and Qadeer 2009]. This work shows that efficient invariant inference can be made possible for invariants of more complex syntactic forms through properties of the transition system and the invariant. Our results go beyond the recent upper bound of Feldman et al. [2020] for maximal systems in that in our results, the diameter need not be one, not all states in the invariant need to be k-reachable, and the invariant can be richer than monotone DNF.

Overall, this paper makes the following contributions:

- We introduce the fence condition, relating a geometric aspect of the invariant to reachability in the system (§4).
- We prove that when the fence condition holds, an interpolation-based algorithm can efficiently infer short monotone DNF invariants (§5). A dual algorithm infers short (anti)monotone CNF invariants. This is the first result showing a polynomial upper bound on the complexity of an interpolation-based invariant inference algorithm with access to a SAT solver as an oracle.
- We show that an algorithm, based on the work by Bshouty [1995], can efficiently infer short almost-monotone DNF invariants when the fence condition holds (§6). A dual algorithm infers short almost-(anti)monotone CNF invariants.
- We show that these results can be obtained by a translation of algorithms from exact concept learning admitting special conditions, made possible by the fence condition (§7).
- We study the robustness of the fence condition under program transformations (§8).

2 OVERVIEW

Suppose we would like to automatically prove the safety of the transition system in Fig. 3. In this simple example, there are n propositional variables $x_1, \ldots, x_n$, and J is a subset of the variables so that $x_1 \in J$. The transition system starts with all variables *true*, and in each iteration either (1) moves a *false* value from one $x_i \in J$ to a different $x_j \in J$ that is not already *false*, (2) turns two $x_i, x_j \in J$ that are *false* to *true*, or (3) assigns arbitrary *true*/*false* values to all variables not in J. The safety property is that the system can never reach the state where $x_1 = false$ and $x_2, \ldots, x_n$ are *true*. Given the transition system and safety property, we would like to find an inductive invariant that establishes it. One inductive invariant here is that the variables in J are always *true*:

$$I = \bigwedge_{x_i \in J} x_i. \tag{1}$$

(For other variables, $x_i \notin J$, this is not true, as x_i can become *false* in havoc_others.)

In invariant inference, the goal is to find such inductive invariants completely automatically. The main motivation of our investigation is the ***theoretical understanding of when invariant inference algorithms are successful***. To this end, we study the complexity of invariant inference algorithms stemming from the seminal ***interpolation-based approach to inference*** by McMillan [2003]. Our formal setting is the problem of finding invariants for propositional transition systems, a fundamental setting which is also relevant for infinite-state systems through predicate abstraction [Flanagan and Qadeer 2002; Graf and Saïdi 1997].

2.1 Interpolation-Based Invariant Inference

We start our investigation with the pioneering interpolation-based algorithm [McMillan 2003], depicted in Alg. 1. The algorithm operates by forward-exploration, in each iteration weakening (adding more states) to the current candidate I, starting from $I = Init$. The algorithm chooses an unrolling bound k, and asserts that I cannot reach *Bad* in k steps (line 4). If the check fails, I is too weak—it includes states that cannot be part of any inductive invariant—and the algorithm restarts, with a larger unrolling bound. (Unless this happens already when $I = Init$, indicating that the system is unsafe; we omit this case here for brevity.) Otherwise, the algorithm computes an *interpolant* χ: a formula ranging over program states that (1) overapproximates the post-image of I, namely, includes all states that are reachable in one step from I; and (2) does not include any state that can reach *Bad* in $k - 1$ steps. In the original work, χ is obtained from a Craig interpolant [Craig 1957] of a bounded model checking (BMC) query, which can be computed from the SAT solver's proof [McMillan 2003]. The candidate invariant is weakened by taking a disjunction with the

interpolant. In this way, each iteration adds to the invariant at least all the states that are reachable in one step from the current candidate but are not part of it (which are counterexamples to its induction), guided by BMC.

Algorithm 1	**Algorithm 2**
Interpolation-based invariant inference	Interpolation by term minimization
[McMillan 2003]	[Bjørner et al. 2013; Chockler et al. 2012]

```
1: procedure Itp-Inference(Init, δ, Bad, k)
2:     I ← Init
3:     while I not inductive do
4:         if δ^≤k(I) ∩ Bad ≠ ∅ then
5:             restart with larger k
6:         find χ such that δ(I) ⟹ χ, δ^≤k−1(χ) ∩ Bad = ∅
7:         I ← I ∨ χ
8:     return I
```

```
1: procedure TermMinItp(I, δ, Bad, k − 1)
2:     χ ← false
3:     while δ(I) ⟹̸ χ do
4:         let σ′ such that σ′ ∈ δ(I), σ′ ⊭ χ
5:         if δ^≤k−1(σ′) ∩ Bad ≠ ∅ then fail
           d ← cube(σ′)
6:         for ℓ in d do
7:             if δ^≤k−1(d \ {ℓ}) ∩ Bad = ∅ then
8:                 d ← d \ {ℓ}
9:         χ ← χ ∨ d
10:    return χ
```

As the original paper shows, with a sufficiently large k, the algorithm is guaranteed to converge to an inductive invariant. However, this may take exponentially many iterations [McMillan 2003]. From a complexity-theoretic perspective, this guarantee can also be achieved by a simple naive search. Clearly, the answer as to why interpolation-based inference is better lies with the virtue of interpolants. However, each bounded proof may allow several interpolants, all satisfying the requirements from $χ$ above, which nonetheless greatly differ from the perspective of invariant inference. Some may be desired parts of the inductive invariant, others might include also states that can reach *Bad* and should not be used. Still other interpolants are safe, but using them would lead to very slow convergence to an inductive invariant. Choosing "good" interpolants is the problem of *generalization*: how should the algorithm choose interpolants so that it finds an invariant quickly? The present view is that generalization strives to abstract away from irrelevant aspects, which is "heuristic in nature" [McMillan 2018]. Perhaps for this reason, there is currently no theoretical understanding of the efficiency of interpolation-based algorithms. In contrast, we identify certain conditions that facilitate a theoretical complexity analysis of this algorithmic approach.

The challenges in a theoretical complexity analysis of this approach are best understood by considering the (termination) analysis of Alg. 1 by McMillan [2003], which is essentially as follows. Suppose k was increased sufficiently to match the *co-reachability-diameter* of the system: the maximal number of steps required for a state to reach a bad state, if it can reach any bad state.[1] Then *any* choice of interpolant $χ$ as above is a safe overapproximation in the sense that no state in $χ$ can reach *Bad* in any number of steps (even greater than k). The candidate I thus never includes states that can reach *Bad*; put differently, it is below the greatest fixed-point (gfp)—the weakest invariant, consisting of all states that cannot reach *Bad*. Because I is always strictly increasing (becoming strictly weaker) and there is only a finite (albeit exponential) number of strictly increasing formulas, Alg. 1 must converge, and at that point I is an inductive invariant.

From the perspective of *complexity*, this termination analysis has several shortcomings::

(1) **Short invariants**: Are there cases where the algorithm is guaranteed to find invariants that are not the gfp? The gfp captures the set of backwards-reachable state in an exact way, but often this is too costly, and unnecessary. For example, in the system of Fig. 3, the states that can reach *Bad* are the states where there is an odd number of *false* variables in J.

---

[1]McMillan [2003] calls this number the "reverse-depth".

The gfp is thus the invariant saying that the number of *false* variables in J is even, whose minimal DNF representation is exponentially long [Crama and Hammer 2011, Theorem 3.19]. Indeed, invariant inference typically strives to achieve an invariant which is "just right" for establishing the safety property of interest, but this is not reflected by the existing theoretical analysis. Can the algorithm benefit from the existence of an invariant that has a short representation (whether or not it is the gfp)?

(2) **Number of iterations**: If the invariant the algorithm finds has a short representation, does the algorithm find it in a small number of iterations? As an illustration, the algorithm could, in principle, resort to inefficiently enumerating the states in the invariant one by one, even though there are more compact ways to represent the invariant.

(3) **Unrolling depth**: Is it actually necessary to use the BMC bound k that is as large as the co-diameter? For example, in the system of Fig. 3, the co-diameter is $\Theta(n)$.[2] This BMC bound could be prohibitively large for the SAT solver. Worse still, the co-diameter can be exponential in general [McMillan 2003]. Can the algorithm always succeed even when using smaller k?

In this work we develop a ***theoretical analysis of an interpolation-based algorithm*** that addresses all these points. The analysis of this algorithm characterizes the invariants that it finds, the number of iterations until convergence, and how large the unrolling bound must be. In this analysis, the inference problem is "well-behaved" if there exists a *short invariant* of *certain syntactic forms* whose *Hamming-geometric boundary* is backwards k-reachable from *Bad*, where k is a bound which can be *smaller than the co-diameter*. If this is the case, we show algorithms that construct and maintain generalizations in specific ways—connected to *exact learning theory*—which are guaranteed to *find inductive invariants efficiently* with a SAT oracle.

Interpolation by covering and term minimization. The algorithmic scheme of Alg. 1 itself is not amenable to such an analysis, because valid choices for χ range between the exact post-image $\delta(I)$ and the set of states that cannot reach *Bad* in k steps, potentially leading to very different outcomes over the algorithm's run. We thus examine a specific method of interpolant construction displayed in Alg. 2, due to Bjørner et al. [2013]; Chockler et al. [2012] and inspired by IC3/PDR [Bradley 2011; Eén et al. 2011]. The procedure iteratively samples states from the post-image of I that should be added to the interpolant χ. Adding a single state exactly, by disjoining the cube of the state $cube(\sigma')$—the conjunction of all literals that hold in the state—would converge slowly. Instead, the procedure drops literals from $cube(\sigma')$ (thereby including more states) as long as no state that satisfies the remaining conjunction can reach *Bad* in $k-1$ steps,[3] and then disjoins the result to the interpolant. By construction, all the states in χ cannot reach *Bad* in $k-1$ steps, and the procedure terminates when all of $\delta(I)$ is covered by χ. It may seem inefficient to overapproximate from a single state in each iteration, compared to proof-based methods; however, our results show that under certain conditions each such iteration makes significant progress.

2.2 The Complexity of Interpolation-Based Inference

The interpolation-based inference of Alg. 1 with the interpolation procedure of Alg. 2 produce invariants in disjunctive normal form (DNF). (For example, the invariant in Equation (1) is a DNF formula with one term.) If the shortest invariant in DNF is exponentially long, clearly an exponential

---

[2]The state with a maximal odd number of variables in J except x_1 having value *false* requires $\left\lfloor \frac{|J|-2}{2} \right\rfloor$ iterations of `turn_two_off` to turn two such variables to *true* until only one is left *false*, an iteration of `hot_potato` to move it to x_1, and another to turn the variables not in J to *true*.

[3]In practice, this can be made more efficient by first obtaining an *unsat core* of the unsat BMC query, and then explicitly dropping literals one by one. Empirically, Chockler et al. [2012] found that the extra time spent in explicit minimization after the unsat core is compensated by fewer iterations.

number of iterations is necessary. ***Suppose that the system admits a short DNF invariant*** I**.** ***When is the interpolation-based inference algorithm guaranteed to be efficient?*** By efficient we mean that the number of steps the algorithms performs is polynomial in the number of variables and the length of I, where each k-BMC query is counted as a single step of calling a SAT oracle. This is a difficult question, because even if an invariant with a short representation exists, the algorithm might miss it or learn a longer representation. Our solution is based on two ingredients: (1) the *boundary* of the invariant, which a condition we call the *fence* condition ties to reachability, and (2) utilizing the syntactic shape of the invariant, an aspect in which, as we show, ideas from *exact learning* are extremely relevant.

2.2.1 The Boundary of Inductive Invariants. Our algorithm makes decisions using BMC: whether or not a literal is to be dropped, whether or not some states are to be added to the invariant, is a choice made on the basis of bounded reachability information (because unbounded reachability is unknown). This is why the algorithm might fail and restart with a larger k (line 5). The fence condition's guarantees that this would not happen. The idea is to require that BMC finds useful information when it is invoked on sets that have a special role: the states on the **boundary of the inductive invariant** I. The (outer) boundary $\partial^-(I)$ is the set of states σ that do *not* belong to I, but are "almost" in I: flipping just one bit in σ yields a state that *does* belong to I. Put differently, the boundary is the set of states in $\neg I$ that have a Hamming neighbor (Hamming distance 1) in I. For example, the boundary of the invariant in Equation (1) consists of the states where $|\{i \in J \mid x_i = \textit{false}\}| = 1$: such states are not in I, but flipping the single *false* variable in J results in a state that *does* belong to I. (See Fig. 1 for an illustration of the boundary of this invariant with $n = 3$ and $J = \{x_1, x_2, x_3\}$.) Note that not all states in $\neg I$ are on the boundary: states with more than one *false* variable in J also belong to $\neg I$, but their Hamming distance from I is larger than 1.

The **fence condition** requires that the **states in the boundary** $\partial^-(I)$ **reach Bad in at most** k **steps**. For example, in the system of Fig. 3, the invariant in Equation (1) satisfies the fence condition with $k = 2$; in at most two steps every state in its boundary reaches *Bad* (one step to move the *false* value to x_1, and another to turn off variables not in J). This property is key for the interpolation algorithm to successfully and consistently find an invariant. We formally define the boundary and the fence condition in §4. (See Fig. 2 for an illustration.)

It should be noted that the fence condition is a property of a specific invariant. Some invariants may not satisfy the fence condition for a given k or even for any k, depending on whether and how quickly states outside the invariant can reach *Bad*. Intuitively, this reflects natural differences between invariants: some invariants are well suited to be discovered by the algorithm, while others are such that the algorithm might overshoot and miss them in some of its executions.

Every safe system admits an invariant that is k-fenced, because the gfp if k-fenced for k that is the co-diameter (Lemma 4.4). The power of the fence condition is in that it allows to prove convergence even with k that is strictly smaller than the co-diameter, and based on invariants that need not be the gfp, which is an important ingredient of our efficiency results.[4]

2.2.2 Convergence With the Fence Condition and a Sufficient BMC Bound. Intuitively, when the fence condition does not hold, if Alg. 2 discovers a state $\sigma^+ \models I$ and checks whether it can drop a literal on which its Hamming neighbor $\sigma^- \models \neg I$ disagrees, if σ^- does not reach *Bad* in k steps we could include σ^- in the candidate even though it is not in I. In §5.2, we prove that the fence condition ensures that the candidate invariant our algorithm constructs is always below I, and therefore converges to I itself or a stronger invariant, *whichever* counterexamples to induction the solver

---

[4]Even for the co-diameter, the existing analysis by McMillan [2003] does not derive complexity bounds, while we are able to do so using techniques from exact concept learning.

returns, and *regardless* of the order in which Alg. 2 attempts to drop literals (Lemma 5.1)—although, without further assumptions which we explore next, this convergence may happen only after many iterations of the algorithm. But before we elaborate on this, two points merit emphasis:

First, not all the states in $\neg I$ need to reach *Bad* in k steps, and k can be smaller than the co-diameter (which addresses question (3) above). For example, in Fig. 3, our results show that $k = 2$ suffices, whereas the co-diameter is linear in the number of variables. Further, this deviates from the original termination argument above, and facilitates an analysis where the invariant to which the algorithm converges is not the gfp. (The latter point already addresses question (1) above.)

Second, the algorithm does not posses any a-priori knowledge of the invariant I; the existence of an invariant as mandated by the fence condition suffices to guarantee convergence.

We are now ready to tackle the question of *efficient convergence* when I is not only k-fenced but also belongs to syntactic class of "manageable" formulas.

2.2.3 Efficient Inference of Short Monotone Invariants. The fence condition guarantees that the generated interpolants underapproximate an invariant I. How many interpolants must the algorithm find before it converges to an invariant? We prove in §5.3 that if I is in monotone DNF, that is, all variables appear positively, then the number of iterations of Alg. 1 is bounded by the number of terms in I if I is k-fenced (Thm. 5.6)—thus addressing question (2) above. ***Overall, this is a theorem of efficient invariant inference by Alg. 1.*** For example, the system in Fig. 3 admits the short monotone DNF invariant in Equation (1), hence by our results the algorithm efficiently infers an invariant for this system.

An intriguing consequence of formal efficiency results for an algorithm is that when the algorithm fails to converge, this is a ***witness that an invariant of a certain type does not exist***. Thus, if the algorithm continues to execute beyond the number of steps mandated by our upper bound, this means that there is no monotone DNF k-fenced invariant with a specified number of terms. This may indicate a bug rendering the system unsafe, or perhaps that an invariant exists but it is not k-fenced, not monotone, or too long.

We also derive a dual efficiency result: a *dual-interpolation* algorithm achieves efficient inference for short (anti)monotone CNF (Corollary 5.9)—which are CNF invariants where all variables appear negatively—when a *dual fence condition* holds: the inner boundary $\partial^+(I)$ is the set of states in I that have a Hamming neighbor in $\neg I$, and it must be k-reachable from *Init*.

2.3 Efficient Inference Beyond Monotone Invariants and Exact Learning Theory

The interpolation-based algorithm is not guaranteed to perform a small number of iterations when I is not monotone. ***Is provably efficient inference beyond monotone invariants possible?*** In §6, we obtain an efficient inference algorithm for the wider class of *almost-monotone* DNF invariants, which are DNF formulas that have at most $O(1)$ terms that include negative literals, provided that the fence condition is satisfied (Corollary 6.11). This upper bound is achieved by a different, yet related algorithm, which is based on the celebrated work of Bshouty [1995] in machine learning. Roughly, the algorithm uses several instances of Alg. 1 that learn several overapproximations of I, each of them monotone under a different translation of the variables. A dual result holds for almost-(anti)monotone CNF invariants and the dual fence condition (Corollary 6.12).

Underlying this development is the realization that efficient inference based on the fence condition has close connections to efficient *exact concept learning* [Angluin 1987]. In exact concept learning, the goal is to learn an unknown formula φ through a sequence of queries to a teacher. The most prominent types of queries are (1) equivalence, in which the learner suggests a candidate θ, and the teacher returns whether $\theta \equiv \varphi$, or a differentiating counterexample; and (2) membership, in which the learner chooses a valuation v and the teacher responds whether $v \models \varphi$. These queries are hard

to implement in an invariant inference setting [Feldman et al. 2020; Garg et al. 2014]. Nevertheless, we show that the algorithm which achieves Corollary 6.11 can be obtained directly from the exact concept learning algorithm by Bshouty [1995] through a transformation which, *when the fence condition holds*, can implement *certain* equivalence and membership queries using BMC (§7.2). This is surprising because in invariant inference, unlike classical concept learning, the "teacher" (SAT solver) does not know the target "concept" I. (We also provide a self-contained analysis of this algorithm, based on the monotone theory [Bshouty 1995], in §6.1.) It is interesting to note that applying this transformation to the exact concept learning algorithm for monotone DNF formulas by Angluin [1987] produces an algorithm that essentially matches Alg. 1+Alg. 2.

Lastly, we extend our translation from exact learning algorithms to show that *every* exact learning algorithm from equivalence and membership queries can be efficiently implemented, provided that the fence condition and its dual both hold together. This yields an efficient inference algorithm for all invariants that have both a short CNF and a short DNF representation (not necessarily monotonic), including formulas that can be expressed by small Boolean decision trees (Thm. 7.8), when this two-sided fence condition holds.

2.4 Robustness and Non-Robustness

One of the most interesting questions for invariant inference as a practical methodology is robustness: ***How do modifications to the program affect invariant inference?*** On the one hand, it is desirable that if an invariant could be found before, and the program undergoes an "inconsequential" transformation, then the algorithm would successfully find the invariant also after the transformation. On the other, when the algorithm does not manage to find an invariant, altering the program can prove a successful strategy to achieve convergence [Borralleras et al. 2017; Cyphert et al. 2019; Feldman et al. 2019; Namjoshi 2007; Sagiv et al. 2002; Sharma et al. 2011]. The effect of program transformations on inference depends on the inference algorithm.[5] In §8 we study this question from the perspective of the fence condition: if an invariant is k-fenced before the transformation, does this still hold after the transformation (thereby making our efficiency results applicable)? We show simple transformations that are robust, ensuring that the invariant is k-fenced also after the transformation: variable renaming and translation, and strengthening safety properties. We then show that interesting transformations that add new variables are not robust: instrumentation with a derived relation, and "monotonization" of an invariant using new variables that track negations.

For example, suppose we add a bit q to the example of Fig. 3 to represent the parity of the variables in J: the xor $\oplus_{x_i \in J} x_i$. We initialize q to the correct initial value (which is $|J| \pmod 2$). The motivation is to use q to prove that the system avoids the bad state, so we consider an error when $x_1 = false, x_2 = \ldots = x_n = true$, and q correctly matches the parity of the variables in J in the error state (i.e. $q = |J| - 1 \pmod 2$). The parity is not changed by any of the actions, so q is not modified in any action. Under this transformation, Equation (1) is still an inductive invariant, but it is no longer backwards k-fenced (for any k): a state with exactly one variable *true* out of J but with the incorrect parity in q cannot reach the bad state. This suggests that the algorithm cannot be guaranteed to converge to the original invariant. (Indeed, the motivation for the transformation is the different invariant $q = |J| \pmod 2$; alas, in this case, this invariant is also not backwards k-fenced: its boundary consists of all states with $q \neq |J| \pmod 2$, but this includes states with an even number of *true* variables from J, which cannot reach the bad state.)

This non-robustness result matches the way invariant inference behaves in practice and the butterfly effect of introducing a derived relation, which indicates that our theoretical analysis can

---

[5]For example, the inference of conjunctive invariants via Houdini [Flanagan and Leino 2001] is robust, and always finds an invariant if one exists in a polynomial number of SAT calls.

reproduce some realistic phenomena. The non-robustness result means that the algorithm is not guaranteed to converge to an invariant that does not use the new variable; nonetheless, invariants that use the new variable may or may not satisfy the fence condition, and this depends on the example. Also in practice, the inference algorithm learns properties that use the new variable, and the transformation may help inference to converge to a new invariant, but might not.

3 PRELIMINARIES

States, transition systems, inductive invariants. We consider the safety of transition systems defined over a propositional vocabulary $\Sigma = \{p_1, \ldots, p_n\}$ of n Boolean variables. A *state* is a *valuation* to Σ. If x is a state, $x[p]$ is the value (*true* or *false*) that x assigns to the variable p. We write $x[p \mapsto z]$ for the state obtained from x by assigning the value z to p. A *transition system* is a triple $(\textit{Init}, \delta, \textit{Bad})$ where $\textit{Init}, \textit{Bad}$ are formulas over Σ denoting the set of initial and bad states (respectively), and the *transition relation* δ is a formula over $\Sigma \uplus \Sigma'$, where $\Sigma' = \{x' \mid x \in \Sigma\}$ is a copy of the vocabulary used to describe the post-state of a transition. We assume that $\textit{Init} \not\equiv \textit{false}$, $\textit{Bad} \not\equiv \textit{true}, \textit{false}, \textit{Init} \implies \neg \textit{Bad}$. A transition system is *safe* if all the states that are reachable from the initial states via any number of steps of δ satisfy $\neg \textit{Bad}$. An *inductive invariant* is a formula I over Σ such that (i) $\textit{Init} \implies I$, (ii) $I \wedge \delta \implies I'$, and (iii) $I \implies \neg \textit{Bad}$, where I' denotes the result of substituting each $x \in \Sigma$ for $x' \in \Sigma'$ in I, and $\varphi \implies \psi$ denotes the validity of the formula $\varphi \to \psi$. In the context of propositional logic, a transition system is safe if and only if it has an inductive invariant. An *inductiveness check*, typically implemented by a SAT query, returns whether a given candidate φ is an inductive invariant, and a counterexample if it violates one of the implications (i)–(iii) above. When implication (ii) is violated, a *counterexample to induction* is a pair of states σ, σ' such that $\sigma, \sigma' \models I \wedge \delta \wedge \neg I'$ (where the valuation to Σ' is taken from σ').

Bounded reachability and bounded model checking. Given a transition system $(\textit{Init}, \delta, \textit{Bad})$ and a *bound* $k \in \mathbb{N}$, the k-forward reachable states $\delta^{\leq k}(\textit{Init})$ are the states σ such that there is an execution of δ of length at most k starting from a state in $\textit{Init}$ and ending at σ. The k-backwards reachable states $(\delta^{-1})^{\leq k}(\textit{Bad})$ are the states σ such that there is an execution of δ of length at most k starting from σ and ending at a state in *Bad*. *Bounded model checking (BMC)* [Biere et al. 1999] checks whether a set of states described by a formula ψ is

- forwards unreachable, checking $\delta^{\leq k}(\textit{Init}) \cap \psi \overset{?}{=} \emptyset$. As a SAT query, this is implemented by checking whether $\textit{Init}(\Sigma_0) \wedge \bigwedge_{i=0}^{k-1} \delta(\Sigma_i, \Sigma_{i+1}) \wedge \left(\bigvee_{i=0}^{k} \psi(\Sigma_i) \right)$ is unsatisfiable; or

- backwards unreachable, checking $\delta^{\leq k}(\psi) \cap \textit{Bad} \overset{?}{=} \emptyset$. As a SAT query, this is implemented by checking whether $\psi(\Sigma_0) \wedge \bigwedge_{i=0}^{k-1} \delta(\Sigma_i, \Sigma_{i+1}) \wedge \left(\bigvee_{i=0}^{k} \textit{Bad}(\Sigma_i) \right)$ is unsatisfiable.

$(\Sigma_0, \ldots, \Sigma_k$ are distinct copies of the vocabulary Σ.) A k-BMC check is either one, with bound k.

Forwards/backwards duality. A known duality [e.g Feldman et al. 2020, Appendix A] substitutes forwards and backwards reachability: The dual of a transition system $(\textit{Init}, \delta, \textit{Bad})$ is $(\textit{Bad}, \delta^{-1}, \textit{Init})$, where $\delta^{-1} = \delta(\Sigma', \Sigma)$ is the inverse of δ. The dual transition system is safe iff the original one is, and I is an inductive invariant for the original system iff $\neg I$ is an inductive invariant for the dual. Note that the set of k-forwards reachable states in the original system is exactly the set of k-backwards reachable states in the dual system. Given an inference algorithm $\mathcal{A}(\textit{Init}, \delta, \textit{Bad})$, the *dual algorithm* is $\mathcal{A}^*(\textit{Init}, \delta, \textit{Bad}) = \neg \mathcal{A}(\textit{Bad}, \delta^{-1}, \textit{Init})$. Complexity results translate between the algorithm and its dual: $\mathcal{A}^*(\textit{Init}, \delta, \textit{Bad})$ finds an invariant in t SAT calls if $\mathcal{A}(\textit{Bad}, \delta^{-1}, \textit{Init})$ finds an invariant in t SAT calls.

CNF, DNF, and cubes. A *literal* ℓ is a variable p (positive literal) or its negation $\neg p$ (negative literal). A *term* is a conjunction of literals; at times, we also refer to it as a set of literals. A formula is in

disjunctive normal form (DNF) if it is a disjunction of terms. A clause is a disjunction of literals. A formula is in *conjunctive normal norm (CNF)* if it is a conjunction of clauses. The *cube* of a state σ, denoted $cube(\sigma)$, is the term that is the conjunction of all the literals that are satisfied in σ.

Complexity of SAT-based invariant inference. We measure the complexity of a SAT-based inference algorithm by (i) the number of inductiveness checks, (ii) the number of k-BMC checks, and (iii) the number of other steps, when each SAT call is considered one step (an oracle call). The complexity parameters are the number of variables $n = |\Sigma|$ and syntactic measures of the length of the target invariant. The inference of anti/monotone CNF/DNF invariants with n clauses/terms is in general **NP**-hard with access to a SAT solver [Feldman et al. 2020; Lahiri and Qadeer 2009].

4 THE BOUNDARY OF INDUCTIVE INVARIANTS

In this section we present the backwards k-fenced condition, which is the foundation of our analysis of inference algorithms and all our convergence results.

Definition 4.1 (Neighborhood, Boundary). Two states σ_1, σ_2 are *neighbors* if their Hamming distance is one, namely, $|\{p \in \Sigma \mid \sigma_1 \models p, \sigma_2 \not\models p \text{ or } \sigma_1 \not\models p, \sigma_2 \models p\}| = 1$. The *neighborhood* of a state σ, denoted $N(\sigma)$, is the set of neighbors of σ. The *inner-boundary* of a set of states S is $\partial^+(S) = \{\sigma \in S \mid N(\sigma) \cap \bar{S} \neq \emptyset\}$ (where $\bar{S}$ is the complement of S). Note that $\partial^+(S) \subseteq S$, and the inclusion may be strict (when some $\sigma \in S$ has no neighbors outside of S). The *outer-boundary* is $\partial^-(S) = \partial^+(\bar{S})$. That is, $\partial^-(S) = \{\sigma \notin S \mid N(\sigma) \cap S \neq \emptyset\}$.

Definition 4.2 (Backwards k-Fenced). For a transition system $(Init, \delta, Bad)$, an inductive invariant I is *backwards k-fenced* for $k \in \mathbb{N}$ if $\partial^-(I) \subseteq (\delta^{-1})^{\leq k}(Bad)$.

More explicitly, an invariant I is backwards k-fenced if every state in $\neg I$ that has a Hamming neighbor in I can reach Bad in at most k steps.

Example 4.3. Consider a program that manipulates two numbers represented in binary by $\text{x} = x_1, \ldots, x_n$ and $\text{y} = y_1, \ldots, y_n$. Initially, x is *odd* ($x_n = 1$), and y is *even* ($y_n = 0$). In each iteration, y is incremented by an even number and x is incremented by y (all computations are mod 2^n). The bad states are those where x is even. An inductive invariant I states that $odd(\text{x}) \wedge even(\text{y})$.

Every state in $\neg I$—and in particular, in $\partial^-(I)$—reaches a bad state in at most one step. This is because every state where $even(\text{x})$ holds is bad, and every state where $odd(\text{y})$ holds is either bad, or the step that adds y to x leads to a bad state. Hence, I is 1-backwards fenced.

Now consider the same system except there is a flag z that decides whether x is modified; the system takes a step only if $\text{z} = false$. The same invariant from before applies, but it is no longer backwards 1-fenced. This is because the state in $\partial^-(I)$ where x is odd, y is odd, and $\text{z} = false$ cannot reach a bad state (nor perform any transition). However, a different invariant, $odd(\text{x}) \wedge (even(\text{y}) \vee \neg\text{z})$ is 1-backwards fenced in this system.

In every system, this condition holds for at least one inductive invariant and for some finite k: the gfp—the weakest invariant, that allows all states but those that can reach Bad in any number of steps—satisfies the condition with the co-diameter, the number of steps that takes for all states that can reach Bad to do so.

LEMMA 4.4. *Every safe transition system* $TS = (Init, \delta, Bad)$ *admits an inductive invariant* $gfp = \neg\left((\delta^{-1})^{\leq\infty}(Bad)\right)$ *that is backwards k-fenced for k that is the co-diameter: the minimal k such that* $(\delta^{-1})^{\leq k}(Bad) = (\delta^{-1})^{\leq\infty}(Bad)$.[6]

---

[6]All omitted proofs appear in the supplementary materials.

While this lemma shows the existence of a backwards fenced invariant through the gfp and co-diameter, the k-fence condition is more liberal: it can hold also for an invariant when not *every* state in $\neg I$ reaches *Bad* in k steps, and only the states in $\partial^-(I)$ do. An example demonstrating this appears in §2. An additional example follows.

Example 4.5. Consider an example of a (doubly)-linked list traversal, using i to traverse the list backwards, modeled via predicate abstraction following Itzhaky et al. [2014]. The list starts at h. Initially, i points to some location that may or may not be part of the list, and in each step the system goes from i to its predecessor, until that would reach x. We write s $\rightsquigarrow$ r to denote that r is reachable from s by following zero or more links. Consider the initial assumption h $\rightsquigarrow$ x, but i $\not\rightsquigarrow$ x (it may be that x $\rightsquigarrow$ i, or that i is not at all in the list). The bad states are those where i = h.

An inductive invariant for this system is h $\rightsquigarrow$ x $\wedge$ $\neg$i $\rightsquigarrow$ x. In predicate abstraction, we may take the predicates $p_{h,x} = $ h $\rightsquigarrow$ x, $p_{i,x} = $ i $\rightsquigarrow$ x, and write $I = p_{h,x} \wedge \neg p_{i,x}$, which is a DNF invariant with one term. Hence $\neg I \equiv \neg p_{h,x} \vee p_{i,x}$. The outer boundary $\partial^-(I)$ consists of the states (1) $p_{h,x} = \textit{false}, p_{i,x} = \textit{false}$ and (2) $p_{h,x} = \textit{true}, p_{i,x} = \textit{true}$. Both states are in fact bad states under the abstraction: both include a state where i = h, from which x is unreachable (in (1)) or reachable (in (2)). Thus, I is backwards k-fenced for every $k \geq 0$.

In contrast, not all the states in $\neg I$ reach bad states (in particular, I is not the gfp): the state $p_{h,x} = \textit{false}, p_{i,x} = \textit{true}$ abstracts only states where h $\not\rightsquigarrow$ i, and this remains true after going to the predecessor of i. This shows that the fence condition may hold even though I is not the gfp, and not all states in $\neg I$ reach bad states (in k steps or at all).

5   EFFICIENT INTERPOLATION WITH THE FENCE CONDITION

In this section we prove that the interpolation-based invariant inference algorithm that computes interpolants using sampling and term minimization is efficient for short monotone DNF invariants. §5.1 describes the algorithm. §5.2 derives the algorithm's basic properties and its convergence from the fence condition. §5.3 builds on this to obtain the efficiency result.

5.1   Interpolation by Term Minimization

We begin with a formal presentation of the algorithm we will be analyzing in this section, Alg. 3. It is a simplification of the one presented in §2, "merging" the two loops formed when Alg. 1 uses Alg. 2; instead of first computing an overapproximation χ of the post-image of the entire previous candidate and only then disjoining χ, Alg. 3 disjoins the generalization d to the candidate immediately, so the next counterexample to induction may use pre-states from this generalization, rather from the previous candidate. Our results apply equally also to Alg. 1+Alg. 2.

Alg. 3 starts with the candidate invariant $\varphi = \textit{Init}$, which is gradually increased to include more states. In each iteration, the algorithm performs an inductiveness check (lines 3 and 4), implemented by SAT calls, and terminates if an inductive invariant has been found. If a counterexample to induction (σ, σ') exists, the algorithm generates a term d which includes the post-state σ', and disjoins d to φ to obtain the new candidate (line 10). We refer to d as the *generalization* obtained from σ'. Starting with $cube(\sigma')$—the conjunction that exactly captures σ'—the algorithm drops literals as long as no state in d can reach a bad state in k steps or less (line 8). These checks invoke the SAT solver with BMC queries. If σ' itself reaches a bad state in k steps, no invariant weaker than φ exists, and the algorithm restarts with a larger bound k (line 6). The soundness of this algorithm is immediate: φ always includes *Init*, excludes *Bad* (otherwise the algorithm restarts at line 6), and stops when there is no counterexample to induction.

5.2 Interpolation Confined in the Boundary

Algorithm 3 Interpolation-based inference by term minimization

---

```
 1: procedure ITP-Inference-TermMin(Init, δ, Bad, k)
 2:     φ ← Init
 3:     while φ not inductive do
 4:         let σ, σ' ⊨ φ ∧ δ ∧ ¬φ'
 5:         if δ^{≤k}(σ') ∩ Bad ≠ ∅ then
 6:             restart with larger k
             d ← cube(σ')
 7:         for ℓ in d do
 8:             if δ^{≤k}(d \ {ℓ}) ∩ Bad = ∅ then
 9:                 d ← d \ {ℓ}
10:         φ ← φ ∨ d
11:     return I
```

We now show how the fence condition ensures that Alg. 3 does not "overshoot" beyond the inductive invariant when it uses a large enough BMC bound. We use this to derive a termination property, which we use in §5.3 to obtain the efficiency result.

"Not overshooting" beyond I is formalized in the following lemma:

Lemma 5.1. *Let* $(Init, \delta, Bad)$ *be a transition system, I an inductive invariant, and $k \in \mathbb{N}$. If I is backwards k-fenced, then throughout the execution of* ITP-Inference-TermMin$(Init, \delta, Bad, k)$, *every candidate φ is an underapproximation of I, namely, $\varphi \implies I$.*

PROOF. By induction on the algorithm's iterations: initially, $\varphi = Init \implies I$ since I is an inductive invariant; for later iterations, we show that every $d \implies I$. By induction on the iterations in generalization: when it starts, $d = \sigma'$ satisfies this, because $\varphi \implies I$ so $\sigma \models I$, hence also $\sigma' \models I$ because I is inductive. Later, assume there is a point when $d \implies I$ stops holding: the algorithm drops a literal ℓ, obtaining $\tilde{d} = d \setminus \{\ell\}$ where $d \implies I$ but $\tilde{d} \notimplies I$, and the check passes: $\delta^{\leq k}(\tilde{d}) \cap Bad = \emptyset$. Let $\sigma_2 \models \tilde{d} \wedge \neg I$, and let σ_1 a state which differs from σ_2 on the variable in ℓ alone. Necessarily $\sigma_2 \not\models \ell$ (because $\sigma_2 \not\models d$ as $d \implies I$), so $\sigma_1 \models \ell$. The other literals in $\tilde{d}$ are also satisfied by σ_1 because they are satisfied by σ_2 and σ_1, σ_2 do not differ there. Thus $\sigma_1 \models d$ and in particular $\sigma_1 \models I$. We have thus obtained Hamming neighbors σ_1, σ_2 such that $\sigma_1 \models I, \sigma_2 \not\models I$, but $\delta^{\leq k}(\tilde{d}) \cap Bad = \emptyset$ is a contradiction to I being backwards k-fenced, seeing that $\sigma_2 \models \tilde{d}$. □

This lemma implies that when the condition holds, the invariant I acts as a "barrier" from unsafe overgeneralization, and the algorithm does not fail (line 6 does not execute). This holds even though k may be smaller than the co-diameter (see §2.1), as long as there exists an I which is backwards k-fenced. (I is not known to the algorithm.) Thus, in such a case, the algorithm successfully finds an inductive invariant that is an underapproximation of I. (Without further assumptions, this might take exponentially many steps, a challenge which is the focus of §5.3.)

Lemma 5.2. *Let* $(Init, \delta, Bad)$ *be a transition system and $k \in \mathbb{N}$. If there exists an inductive invariant I that is backwards k-fenced, then* ITP-Inference-TermMin$(Init, \delta, Bad, k)$ *successfully (albeit potentially in an exponential number of steps) finds an inductive invariant φ such that $\varphi \implies I$.*

Recalling that the gfp is backwards k-fenced for k which is at most the co-diameter (Lemma 4.4), this yields a completeness result, akin to the completeness result by McMillan [2003].

Corollary 5.3. *Let* $(Init, \delta, Bad)$ *be a safe transition system. Then there is a bound $k \in \mathbb{N}$ such that* ITP-Inference-TermMin$(Init, \delta, Bad, k)$ *successfully finds an inductive invariant (albeit potentially in an exponential number of steps).*

So far we have provided an upper bound on the k needed for convergence which may be smaller than the co-diameter. (Even the gfp can be k-fenced with k that is smaller than the co-diameter!) The real power of the approach, however, lies in the complexity analysis the fence condition facilitates beyond this completeness result, which we carry out in the next sections.

REMARK 5.1. *What happens to Alg. 3 when the fence condition does not hold? Suppose there are two states $\sigma^- \models \neg I$, $\sigma^+ \models I$ that differ in a single variable p_i. The backwards fence condition requires that σ^- reaches Bad in k steps. Suppose that this is violated, and σ^+ is found as the post-state of a counterexample to induction (line 4). Then, should the algorithm attempt to drop the literal corresponding to p_i in line 8, the BMC check would pass: neither σ^+ nor σ^- can reach Bad in k steps. This would lead to inadvertently adding σ^- to the candidate invariant, violating $\varphi \implies I$ (in contrast to the guarantee of Lemma 5.1), which may hinder convergence. σ^+ may happen not to be a possible counterexample to induction in any intermediate iteration and such a problematic scenario could not materialize. We note however that due to generalization, the counterexamples to induction the algorithm finds can be states that are not reachable in a small number of steps, or indeed at all reachable; in essence, the idea behind the fence condition is to ensure that all the discovered counterexamples continue to come from I (Lemma 5.1) even when assuming that any $\sigma^+ \models I$ can be discovered this way.*

5.3 Inference of Monotone Invariants

In this section we prove that Alg. 3 converges in m iterations when a backwards k-fenced, monotone DNF invariant with m terms exists. Monotonicity is essential: even if a short DNF invariant exists, the fence condition guarantees that each iterations learns an underapproximation of the invariant (Lemma 5.1), but exponentially many iterations could be required before the algorithm converges; we show that this cannot happen with monotone DNF invariants.

Definition 5.4 (Monotone DNF). A formula $\psi \in \text{Mon-DNF}_m$ if it is in DNF with m terms, and variables appear only positively.

Equation (1) is an example of a monotone DNF invariant. The importance of monotonicity for our purpose stems from a classical property (see [e.g. Valiant 1984]), concerning *prime implicants*:

Definition 5.5 (Prime Implicant). A term $\wedge s$, where s is a set of literals, is a *prime implicant* of a formula ψ if $(\wedge s) \implies \psi$, but for every $\ell \in s$, $(\wedge (s \setminus \{\ell\})) \not\implies \psi$. It is *non-trivial* if $\wedge s \not\equiv false$.

CLAIM 5.2 (FOLKLORE). *A non-trivial prime implicant of a monotone DNF formula ψ is a term of ψ.*

We now leverage this for the analysis of Alg. 3. The idea is that the generalizations d the algorithm produces (lines 6 to 9) are prime implicants of I, hence each produces a new term of I when I is monotone.

THEOREM 5.6. *Let $(Init, \delta, Bad)$ be a transition system and $k \in \mathbb{N}$. If there is an inductive invariant $I \in \text{Mon-DNF}_m$ that is backwards k-fenced, then ITP-Inference-TermMin$(Init, \delta, Bad, k)$ converges to an inductive invariant in $O(m)$ inductiveness checks, $O(mn)$ k-BMC checks, and $O(mn)$ time.[7]*

PROOF. We first show the generalizations d are prime implicants of I (this holds even when I is not monotone). That $d \implies I$ was established in the proof of Lemma 5.1. Suppose, for the sake of contradiction, that for some literal $\ell \in d$ it holds that $d \setminus \{\ell\} \implies I$ as well. At some point the algorithm attempted to drop ℓ; let $\tilde{d}$ be the term the algorithm considered at that point. Since we only drop literals afterwards, $d \subseteq \tilde{d}$, and hence $d \setminus \{\ell\} \subseteq \tilde{d} \setminus \{\ell\}$; since these are conjunctions, this means that $\tilde{d} \setminus \{\ell\} \implies d \setminus \{\ell\}$. Since no state in I reaches Bad in any number of steps, $\delta^{\leq k}(d \setminus \{\ell\}) \cap Bad = \emptyset$, and in particular $\delta^{\leq k}(\tilde{d} \setminus \{\ell\}) \cap Bad = \emptyset$. But according to this check, the algorithm would have dropped ℓ, which is a contradiction to $\ell \in d$.

We turn to the overall analysis. By Lemma 5.2 the algorithm does not need to restart, and converges to an inductive invariant $\varphi \implies I$. In every iteration the algorithm disjoins to φ a term of

---

[7]As explained in §3, we measure time complexity modulo the SAT solver—each inductiveness and BMC query is counted as one step.

I, because it is a prime implicant of I and I is monotone and by Claim 5.2. Furthermore, each iteration produces a new term of I, since before it, $\sigma' \not\models \varphi$, but $\sigma' \models d$. Thus, after at most m iterations the algorithm must have added to φ all the terms of I, so $I \implies \varphi$. At this point, from Lemma 5.1, $I \equiv \varphi$ and the algorithm terminates. Each iteration performs one inductiveness check in lines 3 and 4, one k-BMC in line 5, and another k-BMC checks in line 8 for each of the n literals. □

REMARK 5.3. *Thm. 5.6 has implications also for systems that do not satisfy its requirements. If the algorithm has not converged to an invariant in the number of steps specified in Thm. 5.6, this is a* witness *that an invariant satisfying the theorem's conditions* does not exist. *This may indicate a bug rendering the system unsafe, or that an invariant exists but is not k-fenced, not monotone, or is too long.*

REMARK 5.4. *What happens when Alg. 3 uses a bound k' that is too small, even though a k-fenced invariant, with $k > k'$, exists? When using the smaller bound k' naively, the algorithm might overgeneralize to beyond the gfp and fail to find an invariant, or converge to a different invariant that does not admit a short representation in an exponential number of steps. The polynomial bound guaranteed from the larger k can be recovered by increasing the bound once the number of steps surpasses a predefined polynomial, or by running all possible bounds in parallel / diagonally until one instance finds an invariant.*

Inference of (anti)monotone CNF invariants. An efficiency result for inferring (anti)monotone *CNF* invariants that satisfy the *forward* fence condition follows, through the duality discussed in §3.

Definition 5.7. A formula $\psi \in \text{Mon-CNF}_m$ if it is in CNF with m clauses, and variables appear only negatively.

Definition 5.8 (Forwards k-Fenced). For a transition system $(Init, \delta, Bad)$, an inductive invariant I is *forwards k-fenced* for $k \in \mathbb{N}$ if $\partial^+(I) \subseteq \delta^{\leq k}(Init)$.

More explicitly, an invariant I is forwards k-fenced if every state in I that has a Hamming neighbor in $\neg I$ is reachable from $Init$ in at most k steps. From Thm. 5.6 we obtain:

COROLLARY 5.9. *Let $(Init, \delta, Bad)$ be a transition system and $k \in \mathbb{N}$. If there exists an inductive invariant $I \in \text{Mon-CNF}_m$ that is forwards k-fenced, then* Dual-ITP-Inference-ClauseMin = ITP-Inference-TermMin*$(Init, \delta, Bad, k)$ converges to an inductive invariant in $O(m)$ inductiveness checks, $O(mn)$ k-BMC checks, and $O(mn)$ time.*

The code of the dual algorithm appears in the extended version [Feldman et al. 2021].

REMARK 5.5. *Thm. 5.6 and Corollary 5.9 can be extended to the case of* unate *[Angluin et al. 1993] formulas—DNF/CNF formulas where each variable (separately) appears only negatively or only positively—when the backwards/forwards fence condition holds, using essentially the same proof as in Claim 5.2 and Thm. 5.6.*

Alg. 3 beyond monotone invariants. What happens when we try to apply Alg. 3 to infer invariants that satisfy the fence condition but are not monotone (or unate)? By Lemma 5.1, the algorithm would converge to an inductive invariant—but this make take an exponential number of iterations. In Thm. 5.6 we have shown that each iteration produces a prime implicant of I. Unfortunately, the number of prime implicants can be exponential [Chandra and Markowsky 1978; Sloan et al. 2008]. Worse, Aizenstein and Pitt [1995] have shown, in the context of exact learning, that there are cases when there is a unique, short representation of I as a disjunction of prime implicants, but greedily collecting prime implicants cannot escape sifting through exponentially many additional prime implicants (which are subsumed by the prime implicants in the "right" representation).

6 INFERENCE BEYOND MONOTONE INVARIANTS

In this section we transcend the class of monotone invariants, and obtain efficiency results for inferring inductive invariants in almost-monotone DNF (as well as additional classes that admit a small monotone basis), based on the Λ-learning algorithm by Bshouty [1995]. We first present the invariant inference algorithm and a self-contained proof of efficient convergence relying on the fence condition. In §7.2 we obtain an alternative proof by a transformation that can "simulate" the original algorithm through the fence condition.

Algorithm 4 Invariant inference using a known monotone basis, building on [Bshouty 1995]

```
 1: Assuming a known basis {a_1, ..., a_t} (Def. 6.3)
 2: procedure Λ-Inference(Init, δ, Bad, k)
 3:     H_1, ..., H_t ← false
 4:     while H = ⋀_{i=1}^t H_i not an inductive inductive do
 5:         let σ' s.t. (σ, σ') ⊨ H ∧ δ ∧ ¬H'
 6:             or   σ' ⊨ Init ∧ ¬H
 7:         for i = 1, ..., t do
 8:             if σ' ⊭ H_i then
 9:                 d ← MonGenBmc(σ', a_i, Init, δ, k)
10:                 H_i ← H_i ∨ d
11:     return ⋀_{i=1}^t H_i

12: procedure MonGenBmc(σ, a, δ, Bad, k)
13:     if δ^{≤k}(σ) ∩ Bad ≠ ∅ then
14:         restart with larger k
15:     v ← σ; walked ← true
16:     while walked do
17:         walked ← false
18:         for j = 1, ..., n such that a[p_j] ≠ v[p_j] do
19:             x ← v[p_j ↦ a[p_j]]
20:             if δ^{≤k}(x) ∩ Bad = ∅ then
21:                 v ← x; walked ← true
22:     return cube_a(v)
```

6.1 Inference with a Monotone Basis

In this section we present Alg. 4, an algorithm for inferring inductive invariants with a known monotone basis. Choosing an appropriate basis produces the algorithm for almost-monotone DNF, and is described below in §6.2. We use, and present along the way, notions from the monotone theory [Bshouty 1995].[8] The main theme is *variable translation* [see e.g. Wiedemann 1987], which intuitively replaces a variable with its negation. Clearly, a unate formula (a formula where each variable appears either only negatively or only positively—see Remark 5.5) can be made monotone in this way. The key to transcending unate formulas is to consider *multiple* translations and the "monotonization" of the formula according to each of these.

Definition 6.1 (Translation). A *translation* is represented by a valuation a. Intuitively, if $a[p_i] = true$ then the translation defined by a replaces p_i with its negation; if $a[p_i] = false$, it does not. (Semantically, applying a to a state x yields the state obtained by bitwise xor, $x \oplus a$.)

Definition 6.2 (a-Monotonicity). Let v, x be valuations (states) and a a translation. We define the partial order $v \leq_a x$ to mean that x disagrees with a on all variables on which v disagree with a. That is, for every variable p_i such that $v[p_i] \neq a[p_i]$ also $x[p_i] \neq a[p_i]$. (For example, for $a = \overline{0}$, the translation that assigns *false* to all variables, $v \leq_{\overline{0}} x$ means that in each variable where v is *true*, so is x.) A formula ψ is a-monotone if $\forall v \leq_a x. \ v \models \psi$ implies $x \models \psi$.

For example, monotone formulas are $\overline{0}$-monotone. When ψ is in DNF, ψ is a-monotone if every variable p appears in ψ in polarity according to $\neg a[p]$: p appears only positively if $a[p] = false$ and only negatively if $a[p] = true$. In other words, ψ in DNF is a-monotone if interchanging $p, \neg p$ whenever $a[p] = true$ results in a formula that is monotone DNF (per the standard definition).

In general, a formula may not be a-monotone for any a; but it can always be expressed as a conjunction of formulas, each monotone w.r.t. some translation. A set of translations that suffices for this is called a basis:

---

[8]Our presentation differs somewhat from the original. Omitted proofs appear in the supplementary materials.

Definition 6.3 (Monotone Basis). A *basis* is a set of translations $\{a_1, \ldots, a_t\}$. It is a basis *for a formula ψ* if there exist clauses $c_1, \ldots, c_s$ such that $\psi \equiv c_1 \wedge \ldots \wedge c_s$ and for every $1 \leq i \leq s$ there exists $1 \leq j \leq t$ such that c_i is a_j-monotone. That is, in some CNF representation of ψ, every clause is monotone w.r.t. at least one translation from the basis.

Alg. 4 infers an invariant under the assumption that a known set of translations is a basis for the invariant. (We address the choice of the basis in §6.2.) The main idea of Alg. 4 is to think about the desired invariant I as a conjunction $\bigwedge_{i=1}^{t} H_i$ where each H_i is a_i-monotone, and infer the H_i formulas. It is immediate from the definition of a basis that a formula can be expressed in this way by letting each conjunct H_i be the CNF formula that consists of the a_i-monotone clauses in the CNF representation. However, the number of clauses in the CNF representation, s, may be much larger than the number of elements in the basis, t. It is not immediate that there exist *short* a_i-monotone formulas H_i that achieve this, and that they can in fact be inferred efficiently. The main concept that enables this is the *least a_i-monotone overapproximation* of a formula, which the algorithm infers in DNF (and not CNF), resulting in a complexity that depends on t, but not on s.

Given a formula φ and a translation a, the *least a-monotone overapproximation* of φ is the function[9] $\mathcal{M}_a(\varphi)$ defined by

$$x \models \mathcal{M}_a(\varphi) \text{ iff } \exists v.\ v \leq_a x \wedge v \models \varphi.$$

This is the least a-monotone function such that $\varphi \implies \mathcal{M}_a(\varphi)$.

Each H_i will attempt to learn $\mathcal{M}_{a_i}(I)$. The following lemma shows that when this is achieved for all the a_i's in the basis, the original formula is indeed their conjunction:

LEMMA 6.4 (BSHOUTY [1995], LEMMA 3). *If $\{a_1, \ldots, a_t\}$ is a basis for φ, then $\varphi \equiv \bigwedge_{i=1}^{t} \mathcal{M}_{a_i}(\varphi)$.*

A key feature of the $\mathcal{M}_{a_i}(\varphi)$'s is that they are guaranteed to have a short DNF representation, provided that φ has one. This is established by the following lemmas.

LEMMA 6.5 (BSHOUTY [1995], LEMMA 1(7)). *For a term t, $\mathcal{M}_a(t) \equiv \bigwedge \{\ell \in t \mid a \not\models \ell\}$.*

LEMMA 6.6 (BSHOUTY [1995], LEMMA 1(7)). *Let $\varphi = t_1 \vee \ldots \vee t_m$ in DNF. Then the monotonization $\mathcal{M}_a(\varphi) \equiv \mathcal{M}_a(t_1) \vee \ldots \mathcal{M}_a(t_n)$ which is a DNF with m terms.*

Our goal now is to gradually infer $\mathcal{M}_{a_i}(I)$, despite I being unknown. This is done by iteratively obtaining states that ought to be added to the current hypothesis $\bigwedge_{i=1}^{t} H_i$ (line 6). Such a state must be added to every H_i that does not yet include it. This is done by adding (disjoining) an a_i-monotone term to H_i. The term we add (ignoring generalization at this point) is the monotone cube:

Definition 6.7 (Monotone Cube). For a state v and a translation a, the *monotone cube* of v w.r.t. a is the conjunction of all a-monotone literals that hold in v:

$$cube_a(v) = \bigwedge \{p_i \mid v[p_i] = true, v[p_i] \neq a[p_i]\} \wedge \bigwedge \{\neg p_i \mid v[p_i] = false, v[p_i] \neq a[p_i]\}.$$

(In fact, $cube_a(v) \equiv \mathcal{M}_a(cube(v))$, as can be easily seen from Lemma 6.5.)

The monotone cube includes more states than the original, but the following lemma shows that this cannot overgeneralize beyond $\mathcal{M}_{a_i}(\varphi)$:

LEMMA 6.8. *If $v \models \varphi$ then $cube_a(v) \implies \mathcal{M}_a(\varphi)$.*

---

[9]Every Boolean function is expressible by a propositional formula, so we use the function interchangeably with a formula representing it (chosen arbitrarily).

Adding the monotone cube is thus "safe", but may converge to $\mathcal{M}_{a_i}(I)$ too slowly. Lemma 6.6 guarantees the existence of a short DNF representation, but the monotone cube might be too large (include too few states) and not be a term in this representation. To achieve fast convergence we want to learn an actual, syntactic, term of $\mathcal{M}_{a_i}(I)$ whenever we add a term to H_i. The mechanism that produces such terms is *generalization*, in line 20, by means of minimization. The idea is that the more literals on which the state v we will be adding to H_i agrees with a_i, the smaller the conjunction in the $cube_{a_i}(v)$ is. In fact, this minimization successfully achieves an actual term of $\mathcal{M}_{a_i}(I)$. This is established in the next lemma, akin to Claim 5.2 in the purely monotone case.

LEMMA 6.9 (BSHOUTY [1995], PROPOSITION A + LEMMA 1(1)). *A state x is a-minimal positive for φ if $x \models \varphi$ and for every i such that $x[p_i] \neq a[p_i]$ it holds $x[p_i \mapsto a[p_i]] \not\models \varphi$. Let x be a-minimal positive for a formula φ in DNF. Then there is a term t of φ such that $\mathcal{M}_a(t) \equiv cube_a(x)$.*

PROOF. Since $x \models \varphi$ which is in DNF, there is a term t of φ such that $x \models t$. From Lemma 6.8, $cube_a(x) \implies \mathcal{M}_a(t)$. For the other direction, by Lemma 6.5, we need to show that the conjunction $\mathcal{M}_a(t)$ includes all the conjuncts in $cube_a(x)$. To this end, let p_i be such that $x[p_i] \neq a[p_i]$; we need to show that p_i is a literal of t if $x[p_i] = true$, and $\neg p_i$ is a literal of t if $x[p_i] = false$. Suppose otherwise. Then $x[p_i \mapsto \neg x[p_i]]$ also satisfies t. But then $x[p_i \mapsto a[p_i]] = x[p_i \mapsto \neg x[p_i]] \models \varphi$, in contradiction to the premise. □

The algorithm: Λ-Inference. We now collect the ideas from above and describe the algorithm (Alg. 4) for inferring invariants that admit a known monotone basis, based on the backwards fence condition. Assuming a known basis $\{a_1, \ldots, a_t\}$ for the target (unknown) invariant I, the algorithm maintains a sequence $H_1, \ldots, H_t$, where each H_i is an a_i-monotone DNF formula. Each H_i is gradually increased until it is $\mathcal{M}_{a_i}(I)$ (unless an invariant is found earlier). When each H_i attains this limit, $I \equiv \bigwedge_{i=1}^{t} H_i$ and we are done. In a sense, the algorithm combines multiple instances of the inference procedure appropriate for the monotone case (Alg. 3), each for learning an a_i-monotonization of I.

Each H_i starts from *false*. When a state that ought to be added to the current hypothesis $\bigwedge_{i=1}^{t} H_i$ is found (line 6), each H_i that does not include it is increased by adding a new term. In order to learn an actual, syntactic term of $\mathcal{M}_{a_i}(I)$, the algorithm gradually flips bits in the state that disagree with a_i, and *heuristically* checks whether the new state should still be included in the invariant by performing *bounded model checking* (line 20). When the fence condition holds, this mimics the procedure from Lemma 6.9; this is important for the algorithm's efficiency, below.

Before embarking on efficiency guarantees of this algorithm, we note that the algorithm is always sound (even when I is not k-fenced or $\{a_1, \ldots, a_t\}$ is not a basis for I), because it checks that H is inductive before returning; if H not inductive, the algorithm continues to increase H_i's until H includes a state that reaches *Bad* and the algorithm reaches failure.

Our main theorem for this algorithm is that when the k-fenced condition holds, the algorithm can *efficiently* learn every formula for which $\{a_1, \ldots, a_t\}$ is a basis:

THEOREM 6.10. *Let $(Init, \delta, Bad)$ be a transition system, and $k \in \mathbb{N}$. If there exists an inductive invariant I that is backwards k-fenced, $I \in \mathrm{DNF}_m$, and $\{a_1, \ldots, a_t\}$ is a monotone basis for I (Def. 6.3), then Λ-Inference$(Init, \delta, Bad, k)$ converges to an inductive invariant in $O(m \cdot t)$ inductiveness checks, $O(m \cdot t \cdot n^2)$ k-BMC checks, and $O(m \cdot t \cdot n^2)$ time.*

PROOF. Our main claim is that $H_i \implies \mathcal{M}_{a_i}(I)$ and H_i is a_i-monotone (for every i). From this it would follow that $\bigwedge_{i=1}^{t} H_i \implies I$, because $I = \bigwedge_{i=1}^{t} \mathcal{M}_{a_i}(I)$ from the basis assumption and Lemma 6.4. From this it would follow that the counterexample is always positive, $\sigma' \models I$. This implies that $\delta^{\leq k}(\sigma') \cap Bad = \emptyset$, so the algorithm does not fail (line 14).

Initially, the claim holds trivially. Consider an iteration. From the induction hypothesis, and as above, $\sigma' \models I$. Thus generalization begins with $x = \sigma' \models I$. We argue by induction on the steps of generalization that $x \models I$. In each step, we move from a state x to a Hamming neighbor state x' s.t. $\delta^{\leq k}(x') \cap Bad = \emptyset$. By the induction hypothesis and the premise that I is k-backwards fenced, also $x' \models I$, which concludes this induction. The final x in generalization thus has $x \models I$. Therefore, by Lemma 6.8, $cube_{a_i}(x) \implies \mathcal{M}_{a_i}(I)$ for every i. The claim follows.

It remains to argue that after at most $m \cdot t$ iterations the algorithm converges to $\bigwedge_{i=1}^{t} H_i \equiv I$ (unless it terminates earlier with an inductive invariant), because every call to generalization takes at most $O(n^2)$ k-BMC queries. Indeed, every iteration adds at least one term to at least one H_i. For the x that produces the term, $x \models I$, as above, but for every p where $x[p] \neq a[p]$ we have $\delta^{\leq k}(x[p \mapsto a[p]]) \cap Bad \neq \emptyset$, and in particular $x[p \mapsto a[p]] \not\models I$. Using Lemma 6.9, $cube_{a_i}(x)$ is a term of the DNF representation of $\mathcal{M}_{a_i}(I)$ from Lemma 6.6. By Lemma 6.6 this representation has m terms. Overall we need at most $t \cdot m$ iterations. $\qquad\square$

6.2 Choosing a Monotone Basis

Some important classes of formulas have a known basis that the algorithm can use. The class of *r-almost-monotone DNF* is the class of DNF formulas with at most r terms which include negative literals. The set of all translations with at most r variables assigned *true* is a basis for this class [Bshouty 1995]. When $r = O(1)$, the size of this basis is polynomial in n. This is a basis because, when converting an almost-monotone DNF formula to CNF form, every clause has at most r negative literals, which is a-monotone for the translation a which assigns *true* to these variables only. Another interesting class with a known base of size polynomial in n is the class of (arbitrary) DNF formulas with $O(\log n)$ terms, although the construction is less elementary [Bshouty 1995].

Applying Thm. 6.10 with the known basis for r-almost-monotone DNF yields:

COROLLARY 6.11. *Let $(Init, \delta, Bad)$ be a transition system, $k \in \mathbb{N}$, and $r = O(1)$. If there exists an inductive invariant I that is backwards k-fenced, and I is r-almost-monotone DNF with m terms, then Λ-Inference($Init, \delta, Bad, k$) with an appropriate basis converges to an inductive invariant in $poly(m \cdot n)$ inductiveness checks, $poly(m \cdot n)$ k-BMC checks, and $poly(m \cdot n)$ time.*

A dual result for r-almost (anti)monotone CNF invariants, which are CNF formulas with at most r clauses that include positive literals, is as follows:

COROLLARY 6.12. *Let $(Init, \delta, Bad)$ be a transition system, $k \in \mathbb{N}$, and $r = O(1)$. If there exists an inductive invariant I that is forwards k-fenced, I is r-almost-(anti)monotone CNF with m clauses, then Λ-Inference**(Init, \delta, Bad, k)* with an appropriate basis converges to an inductive invariant in $poly(m \cdot n)$ inductiveness checks, $poly(m \cdot n)$ k-BMC checks, and $poly(m \cdot n)$ time.*

7 FROM EXACT LEARNING TO INVARIANT INFERENCE VIA THE FENCE CONDITION

Exact learning with queries [Angluin 1987] is one of the fundamental fields of theoretical machine learning. In this section we show how efficient inference based on the fence condition can be understood as a manifestation of special forms of exact learning algorithms. In §7.2 we obtain Alg. 4 and an algorithm resembling Alg. 3 by a translation from exact learning algorithms that satisfy certain restrictions. In particular, this provides an alternative proof of Thm. 6.10. In §7.3 we show that when both the backwards *and* the forwards fence condition hold, then *every* algorithm for exact learning from equivalence and membership queries can be transformed to an inference algorithm. In particular, this proves that formulas that admit both a short CNF and a short DNF representation (neither necessarily monotone) can be inferred efficiently when the two-sided fence condition holds

(Thm. 7.8). These transformations implement the learning algorithm's queries even though the target invariants are not known to the algorithm or to the SAT solver. Such transformations are impossible in general [Feldman et al. 2020], and here rely on the fence condition.

7.1 Background: Exact Concept Learning with Queries

We begin with some background on exact concept learning with queries. In *exact concept learning* [Angluin 1987], the algorithm's task is to identify an unknown formula φ using queries it poses to a *teacher*. The most studied queries are:

- *Membership*: The algorithm *chooses* a state σ, and the teacher answers whether $\sigma \models \varphi$; and
- *Equivalence*: The algorithm *chooses* a candidate θ, and the teacher returns true if $\theta \equiv \varphi$ or a differentiating counterexample otherwise: a σ s.t. $\sigma \not\models \theta, \sigma \models \varphi$ or $\sigma \models \theta, \sigma \not\models \varphi$.

The question studied is how many equivalence and membership queries suffice to correctly identify an unknown φ from a certain (syntactical) class.

7.2 Inference From One-Sided Fence and Exact Learning With Restricted Queries

The challenge in harnessing exact learning algorithms for invariant inference is the need to also implement the teacher, which is problematic because the algorithm does not know any inductive invariant in advance [Garg et al. 2014]. In this section we overcome this problem using the fence condition, provided that the learning algorithm satisfies some conditions.

Membership queries to an (unknown) target invariant are in general impossible to implement [Feldman et al. 2020; Garg et al. 2014]. Even if we target the clearly-defined gfp specifically, then the query amounts to asking whether σ can reach *Bad* in an unbounded number of steps, but this question is not an easier than the safety problem. If the desired I is not the gfp (say, because the gfp is a complex formula), then it is even less clear how to answer the query.

Equivalence queries are also hard to implement in general [Feldman et al. 2020]: while we can determine inductiveness or find a counterexample, this may be counterexample to induction (σ, σ'), which is a transition, not a single state; deciding whether to return to the learner σ or σ' as a differentiating example depends on whether $\sigma \models I$, which has all the problems of a membership query above. We will circumvent the problem of equivalence queries by considering algorithms that query only on candidates which are underapproximations of the target I:

LEMMA 7.1 (IMPLEMENTING POSITIVE EQUIVALENCE QUERIES). *Let* $(Init, \delta, Bad)$ *be a transition system and I an inductive invariant. Given θ such that $\theta \implies I$, it is possible to decide whether θ is an inductive invariant or provide a counterexample $\sigma \models I, \sigma \not\models \theta$, by*

- *checking whether there is a counterexample $\sigma' \models Init, \sigma' \not\models \theta$ and returning σ' if one exists; and*
- *checking whether there is a counterexample $(\sigma, \sigma') \models \theta \wedge \delta \wedge \neg\theta'$, and returning σ' if one exists.*

Otherwise, θ is an inductive invariant.

Note that $\theta \not\equiv I$ could be an inductive invariant, which does not amount to an equivalence query *per se*, but then the algorithm has already found an inductive invariant and can stop.

Our main observation here is about implementing *membership* queries: that if the fence condition holds for I, then it is possible to efficiently implement *restricted versions* of membership queries:

LEMMA 7.2 (IMPLEMENTING POSITIVE-ADJACENT MEMBERSHIP QUERIES). *Let* $(Init, \delta, Bad)$ *be a transition system and I an inductive invariant that is backwards k-fenced. Given σ s.t. $\sigma \models I$ or $\sigma \in \partial^-(I)$, it is possible to decide whether $\sigma \models I$ using a single k-BMC check of whether $\delta^{\leq k}(\sigma) \cap Bad = \emptyset$.*

PROOF. If $\sigma \models I$, it cannot reach *Bad* in any number of steps, k in particular, and we correctly return *true*. Otherwise, from the premise, $\sigma \in \partial^-(I)$ but $\sigma \not\models I$, so, by the fence condition, we must have $\delta^{\leq k}(\sigma) \cap Bad \neq \emptyset$, and we correctly return *false*.   □

A learning algorithm that only performs such queries induces an invariant inference algorithm.

COROLLARY 7.3. *Let C be a class of formulas. Let $\mathcal{A}$ be an exact concept learning algorithm that can identify every $\varphi \in C$ in at most s_1 equivalence queries and s_2 membership queries. Assume further that when $\mathcal{A}$ performs an equivalence query on θ, always $\theta \implies \varphi$, and when $\mathcal{A}$ performs a membership query on σ, always $\sigma \models \varphi$ or $\sigma \in \partial^-(\varphi)$. Then there exists an invariant inference algorithm that is sound (returns only correct invariants), and, furthermore, can find an inductive invariant for every transition system that admits an inductive invariant $I \in C$ that is backwards k-fenced using at most $s_1 + 1$ inductiveness checks and s_2 k-BMC checks.*

PROOF. We simulate $\mathcal{A}$ using the equivalence queries from Lemma 7.1 and the membership queries from Lemma 7.2. If the fence condition holds, we answer all queries correctly, perhaps except for an equivalence query on θ returning *true* although $\theta \not\equiv I$, but then we have already found an inductive invariant θ and can stop. An additional inductiveness check is used before an invariant is returned to ensure that the result is a correct inductive invariant even when the fence condition does not hold. If the latter inductiveness check fails, the algorithm returns "failure". □

Note that the resulting algorithm is sound even when the fence condition does not hold, although successful and efficient convergence is not guaranteed in this case.

Alg. 3 and exact learning. The interpolation algorithm of Alg. 3 and its efficiency result (Thm. 5.6) can almost exactly be obtained by the transformation of Corollary 7.3 from the exact learning algorithm for monotone DNF [Angluin 1987]. (The code of the exact learning algorithm appears in the extended version [Feldman et al. 2021].) The primary difference between Alg. 3 and the algorithm resulting from the translation is that in the resulting algorithm, generalization starts with the positive literals, filtering the negative literals in advance (as opposed to line 6 in Alg. 3); this is reasonable for searching for monotone invariants, but may not be complete the same way the standard algorithm is (see Corollary 5.3), and also does not extend to unate DNF (see Remark 5.5). Another difference is that the translated algorithm performs BMC on states with more and more *false* entries instead of dropping literals. The similarities outweigh the differences nonetheless.

Alg. 4 and exact learning. The translation in Corollary 7.3 provides an alternative proof of Thm. 6.10.

PROOF OF THM. 6.10. Alg. 4 is obtained by the transformation in Corollary 7.3 applied on the Λ-algorithm for exact concept learning using a known monotone basis by Bshouty [1995, §5] (the algorithm's code appears in the extended version [Feldman et al. 2021]). The bounds on the number of inductiveness and BMC checks in our theorem matches the bounds on equivalence and membership queries of the original algorithm. It remains to argue that the Λ-algorithm satisfies the conditions of Corollary 7.3. Indeed, the hypothesis is always below the true formula and counterexamples are always positive [Bshouty 1995, §5.1.1, inductive property 1], and membership queries are always performed after flipping one bit in a positive example. □

7.3 Inference From Two-Sided Fence and Exact Learning

In this section we simulate arbitrary exact learning algorithms (going beyond the requirements in Corollary 7.3) relying on a *two-sided* fence condition. An important example of such an exact learning algorithm is the CDNF algorithm by Bshouty [1995]. The conditions of the transformation in §7.2 do not hold because this algorithm performs equivalence queries that can return either positive or negative examples. We first exemplify the two-sided fence condition.

Example 7.4. In Fig. 3, the invariant in Equation (1) is backwards 2-fenced (see §2.2.1). It is also 1 *forwards*-fenced: every state in $\partial^+(I) = I$ is reachable in 1 step by havoc_others.

We now show how to implement queries to the invariant using the two-sided fence condition.

LEMMA 7.5 (IMPLEMENTING MEMBERSHIP QUERIES). *Let $(Init, \delta, Bad)$ be a transition system and I an (unknown) inductive invariant that is backwards k_1-fenced and forwards k_2-fenced. Then membership queries to I can be implemented in at most n queries of k_1-BMC and k_2-BMC.*[10]

PROOF. Let σ be a state such that we want to check whether $\sigma \in I$. Choose some known state $\sigma_0 \models Init$, and gradually walk from σ to σ_0, that is, in each step change one variable in σ to match σ_0, i.e. $\sigma \leftarrow \sigma[p \mapsto \sigma_0[p]]$. In each step, check:

- If $\delta^{\leq k_1}(Init) \cap \{\sigma\} \neq \emptyset$ (namely $\sigma \in \delta^{\leq k_1}(Init)$), return *true*. (This is a k_1-BMC check.)
- If $\delta^{\leq k_2}(\sigma) \cap Bad \neq \emptyset$ (namely $\sigma \in (\delta^{-1})^{\leq k_2}(Bad)$), return *false*. (This is a k_2-BMC check.)
- Otherwise, step and recheck.

In this process, when $\sigma \models I$, we cannot return *false*, and when $\sigma \models \neg I$, we cannot return *true*, because I excludes states that can reach Bad (in k_2 steps or more) and $\neg I$ does not include states that are reachable from $Init$ (in k_1 steps or more). If we first cross from I to $\neg I$, we return *true* from I being forwards k_1-fenced. If we first cross from $\neg I$ to I, similarly we return *false* thanks to I being backwards k_2-fenced. Otherwise we reach $\sigma_0 \in I$ without crossing and correctly return *true*. □

An equivalence query can be implemented by an inductiveness check and a membership query, as noted by Feldman et al. [2020]:

LEMMA 7.6 (IMPLEMENTING EQUIVALENCE QUERIES). *Let $(Init, \delta, Bad)$ be a transition system, and I an (unknown) inductive invariant that is forwards k_1-fenced and backwards k_2-fenced. Then given θ it is possible to answer whether θ is an inductive invariant, or provide a counterexample σ such that $\sigma \models \theta, \sigma \not\models I$ or $\sigma \not\models \theta, \sigma \models I$, using an inductiveness check, at most n checks of k_1-BMC and n of k_2-BMC.*

We can use these procedures to implement every exact learning algorithm from (arbitrary) equivalence and membership queries.

COROLLARY 7.7. *Let C be a class of formulas. Let $\mathcal{A}$ be an exact concept learning algorithm that can identify every $\varphi \in C$ in at most s_1 equivalence queries and s_2 membership queries. Then there exists a sound invariant inference algorithm that can find an inductive invariant for every transition system that admits an inductive invariant $I \in C$ that is forwards k_1-fenced and backwards k_2-fenced using at most $s_1 + 1$ inductiveness checks, $n(s_1 + s_2)$ of k_1-BMC checks, and $n(s_1 + s_2)$ of k_2-BMC checks.*

Next, we demonstrate an application of Corollary 7.7 to the inference of a larger class of invariants.

7.3.1 Inference Beyond Almost-Monotone Invariants. Earlier, we have shown that almost-monotone *DNF* invariants are efficiently inferrable when the *backwards* fence condition holds, and similarly for almost-monotone *CNF* when the *forwards* fence condition holds (Corollaries 6.11 and 6.12). We now utilize Corollary 7.7 to the CDNF algorithm by Bshouty [1995] to show that the class of invariants that can be succinctly expressed *both* in DNF and in CNF (not necessarily in an almost-monotone way) can be efficiently inferred when the fence condition holds in *both* directions:

THEOREM 7.8. *There is an algorithm $\mathcal{A}$ that for every input transition system $(Init, \delta, Bad)$ and $k \in \mathbb{N}$, if the system admits an inductive invariant I such that $I \in \text{DNF}_{m_1}$, $I \in \text{CNF}_{m_2}$, and I is both backwards- and forwards- k-fenced, then $\mathcal{A}(Init, \delta, Bad, k)$ converges to an inductive invariant in $O(m_1 \cdot m_2)$ inductiveness checks, $O(m_1 \cdot m_2 \cdot n^3)$ k-BMC checks, and $O(m_1 \cdot m_2 \cdot n^3)$ time.*

---

[10]The proof of this also implies that an invariant that is both forwards k_1-fenced and backwards k_2-fenced is unique, seeing that the implementation of the membership query for both is the same.

As noted by Bshouty [1995], the class of formulas with short DNF and CNF includes the formulas that can be expressed by a small *decision tree*; see the supplementary materials. Similarly, when an r-almost-unate invariant with $O(\log n)$ non-unate variables is fenced both backwards and forwards, it can be inferred by an adaptation of an algorithm by Bshouty [1997]. Whether this is possible based on the one-sided fence condition is an interesting question for future work.

8 ROBUSTNESS AND NON-ROBUSTNESS OF THE FENCE CONDITION

In this section we study the effect of program transformations on the k-fence condition, reflecting on the robustness of guaranteed convergence of the algorithms we study.

Suppose $(Init, \delta, Bad)$ is a transition system that admits an inductive invariant I which is backwards or forwards k-fenced (Defs. 4.2 and 5.8). Suppose that the system is modified in some way to form a new system $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ with an inductive invariant $\widehat{I}$ which is derived from I (often $\widehat{I} = I$ itself). The transformation $\widehat{\cdot}$ is *robust* if $\widehat{I}$ is also backwards/forwards k-fenced in $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$.

We consider several such program transformations and establish their (non)robustness.

8.1 Robustness Under Simple Transformations

8.1.1 Isometries. An isometry with respect to the Hamming distance is a permutation of the variables followed by variable translation per §6.1 [see e.g. Wiedemann 1987]. The new $\widehat{Init}, \widehat{\delta}, \widehat{Bad}, \widehat{I}$ are obtained by renaming the variables according to the permutation, and replacing variables with their negation according to the translation. The isometry preserves the Hamming distance between states. Therefore, if I is backwards/forwards k-fenced, so is $\widehat{I}$.

8.1.2 Conjunctions and Disjunctions. If $(Init, \delta, Bad_1)$ has a backwards (forwards) k-fenced inductive invariant I_1 and similarly $(Init, \delta, Bad_2)$ has I_2, then $I_1 \wedge I_2$ is an inductive invariant for $(Init, \delta, Bad_1 \vee Bad_2)$, and it is also backwards (forwards) k-fenced. Similarly, if $(Init_1, \delta, Bad)$ has a backwards (forwards) k-fenced inductive invariant I_1 and similarly $(Init_1, \delta, Bad)$ has I_2, then $I_1 \vee I_2$ is an inductive invariant for $(Init_1 \vee Init_2, \delta, Bad)$, and it is also backwards (forwards) k-fenced. This is because for any two sets of states S_1, S_2, the boundary satisfies (much like in usual topology)

$$\partial^+(S_1 \cap S_2) \subseteq \partial^+(S_1) \cup \partial^+(S_2) \qquad \partial^-(S_1 \cap S_2) \subseteq \partial^-(S_1) \cup \partial^-(S_2)$$
$$\partial^+(S_1 \cup S_2) \subseteq \partial^+(S_1) \cup \partial^+(S_2) \qquad \partial^-(S_1 \cup S_2) \subseteq \partial^-(S_1) \cup \partial^-(S_2)$$

and backwards (forwards) k-reachability is not reduced in either case.

8.2 Non-Robustness Under Instrumentation

8.2.1 Instrumentation by a Derived Relation. Instrumentation by a derived relation introduces new *ghost variables* that have a defined meaning over the program variables, so as to aid program analysis. (See §9 for a discussion of the role of instrumentation in inference.)

Formally, instrumentation by a derived relation works as follows: The vocabulary is extended with a new variable, yielding $\widehat{\Sigma} = \Sigma \uplus \{q\}$. The transition system is modified to update q according to ψ, its intended meaning: $\widehat{Init} \equiv Init \wedge q \leftrightarrow \psi$, and $\widehat{\delta}$ satisfies $\delta \wedge q \leftrightarrow \psi \implies \widehat{\delta} \wedge q' \leftrightarrow \psi'$ (reading: if q has the correct interpretation in the pre-state, $\widehat{\delta}$ correctly updates it). The bad states $\widehat{Bad}$ are all bad under the old definition ($\widehat{Bad} \implies Bad$), and we expect the projection of $\widehat{Bad}$ to Σ to be exactly Bad. The example instrumentation in §2.4 introduces q to capture $\psi = \oplus_{x_i \in J} x_i$; the transitions there do not modify the truth value of ψ, hence in $\widehat{\delta}$, they do not modify q.

We say that $\widehat{\sigma}$ over $\widehat{\Sigma}$ is *consistent* if $\widehat{\sigma} \models q \leftrightarrow \psi$. Every reachable state of $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ is consistent, but $\partial^+(I)$ includes inconsistent (thus unreachable) states, and this leads to non-robustness. We show (in the extended version) that even if I ranges over the original vocabulary, it cannot be

forwards k-fenced in $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ for any k. For example, the invariant of Equation (1) is forwards 1-fenced before the transformation in §2.4 (see Example 7.4), but not after the instrumentation.

Of course, there is another inductive invariant, $I \wedge q \leftrightarrow \psi$, that we may set as the target for convergence in our analysis of the algorithm. Unfortunately, $\partial^+(I \wedge q \leftrightarrow \psi) = I \wedge q \leftrightarrow \psi$ itself, because flipping q moves to a state outside this invariant. Thus, to satisfy our condition, this entire invariant must be k-reachable, which is a much stronger requirement than I being k-fenced (I must be the least fixed-point and k the diameter). Other k-fenced invariants could also exist.

The discussion in §2.4 demonstrates a similar non-robustness of the backwards fence condition.

8.2.2 Monotonization by Instrumentation. We have shown that the pre-existing invariants are no longer forwards fenced after an instrumentation. If we are interested in an invariant that uses the new variable, the effect is rather idiosyncratic and depends on the new invariant. One class of instrumentations with a clear target for the new invariant is *monotinization by instrumentation*, which is of special importance from the perspective of our results in earlier sections (e.g. Thm. 5.6 and 6.10). Inspired by the importance of the syntactic structure as displayed by our results, it would seem valuable to "monotonize" invariants: introduce new variables capturing the negation of existing variables, and using them to rewrite an invariant I into a *monotone* invariant $\widehat{I}$. For example, suppose that a system admits an inductive invariant $I = (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee \neg p_3)$, in which the variable p_1 appears in I both positively and negatively. We thus introduce a new variable q to be maintained as $\neg p_1$, and write $\widehat{I} = (\neg p_1 \vee \neg p_2) \wedge (\neg q \vee \neg p_3)$, which is (anti)monotone.

Alas, we show that this approach is in general unsuccessful, and does not reduce the inference of fenced invariants from the general case to the monotone case, because $\widehat{I}$ is *not* forwards/backwards k-fenced in $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$, even if I is forwards/backwards k-fenced in $(Init, \delta, Bad)$.

Formally, let q be a variable aiming to capture the negation of a variable $p \in \Sigma$. $\widehat{I}$ is obtained from I by replacing all occurrences of $\neg p$ by q. We modify the system to maintain the relationship between $q, \neg p$: take $\widehat{Init} = Init \wedge p \leftrightarrow \neg q$, and $\widehat{\delta} = p \leftrightarrow \neg q \wedge \delta \wedge p' \leftrightarrow \neg q'$. Note that $\widehat{\delta}$ does not make a step when q is interpreted not as $\neg q$, hence $\widehat{I}$ is indeed an inductive invariant.

We first describe the effect on the forwards fence condition when trying to achieve an (anti)monotone invariant (to match Corollary 5.9). We show that the transformation above does not produce a forwards k-fenced invariant in the case that after replacing occurrences p by $\neg q$, the negative form $\neg p$ must still appear in $\widehat{I}$; roughly, this is when I is not unate w.r.t. p, so using $\neg q$ alone cannot be enough (such as in the example above, in which $\neg p_1$ still appears in $\widehat{I}$).

LEMMA 8.1. *Let $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ be obtained by instrumenting $(Init, \delta, Bad)$ with q capturing $\neg p$, and let I be an inductive invariant for $(Init, \delta, Bad)$ in negation normal form (negations appear only on atomic variables). Let $\widehat{I}$ be obtained by replacing every occurrence of p in I with $\neg q$. Suppose that I is not monotone w.r.t. the variable p: there is σ over Σ such that $\sigma \models I$, $\sigma[p \mapsto true] \models \neg I$. Then $\widehat{I}$ is not forwards k-fenced in $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ for any $k \in \mathbb{N}$.*

The case of the backwards fence condition is dual when $\widehat{Bad} = Bad \wedge p \leftrightarrow \neg q$. If $\widehat{Bad} = Bad$ itself, a similar non-robustness occurs when the critical σ (as in Lemma 8.1) is not in Bad—even though it may reach Bad under δ—because an inconsistent extension of it cannot reach Bad.

9 RELATED WORK

Interpolant generation. Classically, interpolants are generated from unsatisfiability proofs of SAT/SMT solvers, a procedure which has been studied extensively [e.g. Cimatti et al. 2010; Kroening and Weissenbacher 2007; McMillan 2003, 2005, 2011; Vizel et al. 2015a,b]. A relatively recent alternative is to sample states that the interpolant must separate [Sharma et al. 2012]. By iteratively

sampling and combining the resulting interpoloants, it is possible to compute an interpolant of the (entire) formulas [Albarghouthi and McMillan 2013; Bjørner et al. 2013; Chockler et al. 2012; Drews and Albarghouthi 2016]. The interpolation procedure in the algorithm we analyze is from Bjørner et al. [2013]; Chockler et al. [2012], and inspired by IC3/PDR [Bradley 2011; Eén et al. 2011]. It samples from one side—the post image of the previous candidate—and computes an interpolant of the sample and the set of k-backwards reachable states. The EPR interpolation method by Drews and Albarghouthi [2016] is related, with diagrams replacing cubes as the least abstraction of states.

Inference and learning. Ideas from machine learning have inspired many works in invariant inference [e.g. Garg et al. 2014; Jha et al. 2010; Koenig et al. 2020; Sharma and Aiken 2016; Sharma et al. 2013b,a, 2012]. A primary concern has been to overcome the ambiguity of counterexamples to induction, or implication counterexamples, compared to the classical learning settings where examples are labeled positive and negative [Garg et al. 2014]. We resolve this ambiguity through observations of one-sidedness of the algorithm (§7.2) or special invariants to which membership can be implemented (§7.3). Garg et al. [2016] target invariants expressed as decision trees over numerical and Boolean attributes, adapting an algorithm by Quinlan [1986]. The result in §7.3.1 applies to decision trees in the propositional setting, and leverages the CDNF algorithm [Bshouty 1995], which admits a bound on the number of hypotheses the learner presents before converging. The CDNF algorithm [Bshouty 1995] has been applied by Jung et al. [2015] to infer quantified invariants through predicate abstraction. They resolve membership queries by over- and under-approximations to some invariants, and use a random guess when the these are not conclusive, potentially leading to failures and restarts. In this work we use bounded model checking to correctly resolve certain queries under the fence condition, and accomplish efficient overall complexity theorems. Feldman et al. [2020] have shown that membership and equivalence queries cannot be implemented in general in black-box models of invariant inference. In §7.2 we build on the fence condition to implement some forms of queries. Our results in §7.3 generalize the implementability of queries obtained by [Feldman et al. 2020] the (in three aspects as in §1). The CDNF algorithm has also been applied to generate contextual assumptions in assume-guarantee reasoning [Chen et al. 2010].

Complexity of invariant inference. Conjunctive invariants can be inferred in a linear number of SAT calls using Houdini [Flanagan and Leino 2001], and similarly for disjunctive invariants using the dual algorithm [Lahiri and Qadeer 2009]. In richer syntactical classes, which are the domain of algorithms such as interpolation [McMillan 2003] and IC3/PDR [Bradley 2011; Eén et al. 2011], invariant inference becomes much harder [Lahiri and Qadeer 2009], and is **NP**-hard with access to a SAT solver even when the problem is restricted to learning monotone CNF/DNF invariants with a linear number of clauses [Feldman et al. 2020]. These results place no restrictions on the transition system. In this work, the fence condition identifies properties of the transition system and the invariant that evade these hardness results. In the process of showing a separation between ICE learning and algorithms that use richer queries, Feldman et al. [2020] also show a polynomial upper bound on the inference of (anti)monotone CNF invariants. Our results go beyond the class they consider (see §1). The algorithms we study in this paper fall into the black-box model of extended Hoare queries [Feldman et al. 2020]. Whether similar efficiency results can also be obtained in the (non-extended) Hoare-query model—without unrollings for BMC—is interesting for future work.

Robustness of program analysis. The drastic consequences of small program changes on verification tools is sometimes recognized as verification's "butterfly effect" [Leino and Pit-Claudel 2016]. Many program analysis techniques exhibit brittle behaviors [Karpenkov et al. 2016; Logozzo et al. 2014]. This may be in line with the inherent hardness; for instance, the class of programs for which an abstract interpreter is complete is undecidable [Giacobazzi et al. 2015]. A notable exception is the Houdini algorithm, which efficiently finds the strongest conjunctive invariant without any

assumptions on the program [Flanagan and Leino 2001]. In contrast, efficiently inferring invariants from richer syntactic classes is not possible in all programs, because the general case is hard [Feldman et al. 2020; Lahiri and Qadeer 2009], but the fence condition allows efficient inference. As we show, the fence condition is sensitive to some program transformations.

Modifying the program for the sake of verification is emblemed by the practice of ghost code [see e.g. Filliâtre et al. 2016]. Improving invariant inference through program transformations has been applied in many different settings with varying notions of the effect on the underlying inference technology, including improving accuracy, enriching the syntactic search space, and simplifying the quantification structure [Borralleras et al. 2017; Cyphert et al. 2019; Feldman et al. 2019; Namjoshi 2007; Sagiv et al. 2002; Sharma et al. 2011]. Perhaps not surprisingly, there is some empirical evidence that instrumentation can also at times have negative impact on the inference algorithm [Feldman et al. 2019]. Our results suggest that such transformations can have a profound effect on the inference algorithm, in the sense that pre-existing invariants can no longer be the cause of guaranteed convergence after instrumentation by a derived relation.

Learning with errors in queries. Exact learning has also been studied in models where the teacher might not respond "unknown" or answer incorrectly; see [Bshouty 2018] for a short survey. Usually learning is facilitated by bounding the probability of an error or the number of allowed errors. The fence condition gives rise in §7.2 to a different model, which correctly answers all membership queries to examples that are positive or on the concept's outer-boundary, yet can adversarially classify every other example. As we show, several existing algorithms are applicable in this model.

10　CONCLUSION

This work applies ideas from exact concept learning for understanding the behavior of interpolation-based invariant inference, achieving the first polynomial complexity result with access to a SAT oracle for such algorithms. This connection and recent results in inferring quantified invariants and invariants over interesting theories [e.g. Dillig et al. 2013; Feldman et al. 2019; Garg et al. 2016; Gurfinkel et al. 2016; Karbyshev et al. 2017; Koenig et al. 2020; Sharma and Aiken 2016] suggest that there may be an opportunity to develop exact concept learning for infinite domains, which, to the best of our knowledge, is less explored [see e.g. Abouzied et al. 2013; Arias 2004].

The focus of this work in on *worst case* complexity of an interpolation-based invariant inference algorithm, showing conditions that guarantee efficient inference, which is an important step towards a theoretical understanding of invariant inference. We strived for the weakest possible conditions, so that the results would be applicable to as many programs as possible. We believe that the fence condition is the weakest reasonable condition in this setting—we have not found a graceful way to ensure that the bad examples on the boundary are always avoided (see Remark 5.1). We thus believe that to achieve theoretical guarantees for programs that do not satisfy the fence condition, the analysis must go beyond worst case (for example, analyzing convergence with high probability).

ACKNOWLEDGMENTS

REFERENCES

Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. 2013. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*. 49–60. https://doi.org/10.1145/2463664.2465220

Howard Aizenstein and Leonard Pitt. 1995. On The Learnability Of Disjunctive Normal Form Formulas. *Mach. Learn.* 19, 3 (1995), 183–208. https://doi.org/10.1007/BF00996269

Aws Albarghouthi and Kenneth L. McMillan. 2013. Beautiful Interpolants. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. 313–329. https://doi.org/10.1007/978-3-642-39799-8_22

Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2015. Syntax-Guided Synthesis. In *Dependable Software Systems Engineering*. 1–25.

Dana Angluin. 1987. Queries and Concept Learning. *Machine Learning* 2, 4 (1987), 319–342.

Dana Angluin, Lisa Hellerstein, and Marek Karpinski. 1993. Learning Read-Once Formulas with Queries. *J. ACM* 40, 1 (1993), 185–210. https://doi.org/10.1145/138027.138061

Marta Arias. 2004. *Exact learning of first-order horn expressions from queries*. Ph.D. Dissertation. Tufts University, Medford, MA.

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 1999. Symbolic Model Checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*. 193–207. https://doi.org/10.1007/3-540-49059-0_14

Nikolaj Bjørner, Arie Gurfinkel, Konstantin Korovin, and Ori Lahav. 2013. Instantiations, Zippers and EPR Interpolation. In *LPAR 2013, 19th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, December 12-17, 2013, Stellenbosch, South Africa, Short papers proceedings*. 35–41. https://easychair.org/publications/paper/XtN

Cristina Borralleras, Marc Brockschmidt, Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2017. Proving Termination Through Conditional Termination. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*. 99–117.

Aaron R. Bradley. 2011. 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*. 70–87. https://doi.org/10.1007/978-3-642-18275-4_7

Nader H. Bshouty. 1995. Exact Learning Boolean Function via the Monotone Theory. *Inf. Comput.* 123, 1 (1995), 146–153. https://doi.org/10.1006/inco.1995.1164

Nader H. Bshouty. 1997. Simple Learning Algorithms Using Divide and Conquer. *Comput. Complex.* 6, 2 (1997), 174–194. https://doi.org/10.1007/BF01262930

Nader H. Bshouty. 2018. Exact learning from an honest teacher that answers membership queries. *Theor. Comput. Sci.* 733 (2018), 4–43. https://doi.org/10.1016/j.tcs.2018.04.034

Ashok K. Chandra and George Markowsky. 1978. On the number of prime implicants. *Discret. Math.* 24, 1 (1978), 7–11. https://doi.org/10.1016/0012-365X(78)90168-1

Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Ming-Hsien Tsai, Yih-Kuen Tsay, and Bow-Yaw Wang. 2010. Automated Assume-Guarantee Reasoning through Implicit Learning. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*. 511–526. https://doi.org/10.1007/978-3-642-14295-6_44

Hana Chockler, Alexander Ivrii, and Arie Matsliah. 2012. Computing Interpolants without Proofs. In *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*. 72–85. https://doi.org/10.1007/978-3-642-39611-3_12

Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. 2010. Efficient generation of craig interpolants in satisfiability modulo theories. *ACM Trans. Comput. Log.* 12, 1 (2010), 7:1–7:54. https://doi.org/10.1145/1838552.1838559

William Craig. 1957. Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem. *J. Symbolic Logic* 22, 3 (09 1957), 250–268. https://projecteuclid.org:443/euclid.jsl/1183732823

Yves Crama and Peter L. Hammer. 2011. *Boolean Functions - Theory, Algorithms, and Applications*. Encyclopedia of mathematics and its applications, Vol. 142. Cambridge University Press. http://www.cambridge.org/gb/knowledge/isbn/item6222210/?site_locale=en_GB

John Cyphert, Jason Breck, Zachary Kincaid, and Thomas W. Reps. 2019. Refinement of path expressions for static analysis. *Proc. ACM Program. Lang.* 3, POPL (2019), 45:1–45:29. https://doi.org/10.1145/3290358

Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 2013. Inductive invariant generation via abductive inference. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &*

Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013. 443–456.

Samuel Drews and Aws Albarghouthi. 2016. Effectively Propositional Interpolants. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9780)*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, 210–229. https://doi.org/10.1007/978-3-319-41540-6_12

Niklas Eén, Alan Mishchenko, and Robert K. Brayton. 2011. Efficient implementation of property directed reachability. In *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011.* 125–134. http://dl.acm.org/citation.cfm?id=2157675

Grigory Fedyukovich and Rastislav Bodík. 2018. Accelerating Syntax-Guided Invariant Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I.* 251–269. https://doi.org/10.1007/978-3-319-89960-2_14

Yotam M. Y. Feldman, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2020. Complexity and information in invariant inference. *Proc. ACM Program. Lang.* 4, POPL (2020), 5:1–5:29. https://doi.org/10.1145/3371073

Yotam M. Y. Feldman, Mooly Sagiv, Sharon Shoham, and James R. Wilcox. 2021. Learning the Boundary of Inductive Invariants. *CoRR* abs/2008.09909 (2021). https://arxiv.org/abs/2008.09909

Yotam M. Y. Feldman, James R. Wilcox, Sharon Shoham, and Mooly Sagiv. 2019. Inferring Inductive Invariants from Phase Structures. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II.* 405–425. https://doi.org/10.1007/978-3-030-25543-5_23

Jean-Christophe Filliâtre, Léon Gondelman, and Andrei Paskevich. 2016. The spirit of ghost code. *Formal Methods Syst. Des.* 48, 3 (2016), 152–174. https://doi.org/10.1007/s10703-016-0243-x

Cormac Flanagan and K. Rustan M. Leino. 2001. Houdini, an Annotation Assistant for ESC/Java. In *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings.* 500–517.

Cormac Flanagan and Shaz Qadeer. 2002. Predicate abstraction for software verification. In *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002.* 191–202. https://doi.org/10.1145/503272.503291

Pranav Garg, Christof Löding, P Madhusudan, and Daniel Neider. 2014. ICE: A robust framework for learning invariants. In *Computer Aided Verification.* Springer, 69–87.

Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. 2016. Learning invariants using decision trees and implication counterexamples. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016.* 499–512. https://doi.org/10.1145/2837614.2837664

Roberto Giacobazzi, Francesco Logozzo, and Francesco Ranzato. 2015. 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*, Sriram K. Rajamani and David Walker (Eds.). ACM, 261–273. https://doi.org/10.1145/2676726.2676987

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

Arie Gurfinkel, Sharon Shoham, and Yuri Meshman. 2016. SMT-based verification of parameterized systems. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016.* 338–348. https://doi.org/10.1145/2950290.2950330

Shachar Itzhaky, Nikolaj Bjørner, Thomas W. Reps, Mooly Sagiv, and Aditya V. Thakur. 2014. Property-Directed Shape Analysis. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings.* 35–51. https://doi.org/10.1007/978-3-319-08867-9_3

Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010.* 215–224. https://doi.org/10.1145/1806799.1806833

Yungbum Jung, Soonho Kong, Cristina David, Bow-Yaw Wang, and Kwangkeun Yi. 2015. Automatically inferring loop invariants via algorithmic learning. *Math. Struct. Comput. Sci.* 25, 4 (2015), 892–915. https://doi.org/10.1017/S0960129513000078

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

Egor George Karpenkov, David Monniaux, and Philipp Wendler. 2016. Program Analysis with Local Policy Iteration. In *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings.* 127–146. https://doi.org/10.1007/978-3-662-49122-5_6

Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. 2020. First-order quantified separators. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 703–717. https://doi.org/10.1145/3385412.3386018

Daniel Kroening and Georg Weissenbacher. 2007. Lifting Propositional Interpolants to the Word-Level. In *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings.* IEEE Computer Society, 85–89. https://doi.org/10.1109/FAMCAD.2007.13

Shuvendu K. Lahiri and Shaz Qadeer. 2009. Complexity and Algorithms for Monomial and Clausal Predicate Abstraction. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings.* 214–229.

K. Rustan M. Leino and Clément Pit-Claudel. 2016. Trigger Selection Strategies to Stabilize Program Verifiers. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I.* 361–381. https://doi.org/10.1007/978-3-319-41528-4_20

Francesco Logozzo, Shuvendu K. Lahiri, Manuel Fähndrich, and Sam Blackshear. 2014. Verification modulo versions: towards usable verification. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 294–304. https://doi.org/10.1145/2594291.2594326

Kenneth L. McMillan. 2003. Interpolation and SAT-Based Model Checking. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings.* 1–13.

Kenneth L. McMillan. 2005. An interpolating theorem prover. *Theor. Comput. Sci.* 345, 1 (2005), 101–121. https://doi.org/10.1016/j.tcs.2005.07.003

Kenneth L. McMillan. 2006. Lazy Abstraction with Interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings.* 123–136. https://doi.org/10.1007/11817963_14

Kenneth L. McMillan. 2011. Interpolants from Z3 proofs. In *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, Per Bjesse and Anna Slobodová (Eds.). FMCAD Inc., 19–27. http://dl.acm.org/citation.cfm?id=2157661

Kenneth L. McMillan. 2018. Interpolation and Model Checking. In *Handbook of Model Checking.* 421–446. https://doi.org/10.1007/978-3-319-10575-8_14

Kedar S. Namjoshi. 2007. Symmetry and Completeness in the Analysis of Parameterized Systems. In *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings.* 299–313. https://doi.org/10.1007/978-3-540-69738-1_22

Ryan O'Donnell. 2014. *Analysis of Boolean Functions.* Cambridge University Press. http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/analysis-boolean-functions

J. Ross Quinlan. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (1986), 81–106. https://doi.org/10.1023/A:1022643204877

Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. 2002. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.* 24, 3 (2002), 217–298.

Rahul Sharma and Alex Aiken. 2016. From invariant checking to invariant inference using randomized search. *Formal Methods in System Design* 48, 3 (2016), 235–256. https://doi.org/10.1007/s10703-016-0248-5

Rahul Sharma, Isil Dillig, Thomas Dillig, and Alex Aiken. 2011. Simplifying Loop Invariant Generation Using Splitter Predicates. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings.* 703–719. https://doi.org/10.1007/978-3-642-22110-1_57

Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, Percy Liang, and Aditya V. Nori. 2013b. A Data Driven Approach for Algebraic Loop Invariants. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings.* 574–592. https://doi.org/10.1007/978-3-642-37036-6_31

Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, and Aditya V. Nori. 2013a. Verification as Learning Geometric Concepts. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings.* 388–411.

Rahul Sharma, Aditya V. Nori, and Alex Aiken. 2012. Interpolants as Classifiers. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings.* 71–87. https://doi.org/10.1007/978-3-642-31424-7_11

Robert H. Sloan, Balázs Szörényi, and György Turán. 2008. On k-Term DNF with the Largest Number of Prime Implicants. *SIAM J. Discret. Math.* 21, 4 (2008), 987–998. https://doi.org/10.1137/050632026

Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. 2013. Template-based program verification and program synthesis. *STTT* 15, 5-6 (2013), 497–518.

Leslie G. Valiant. 1984. A Theory of the Learnable. *Commun. ACM* 27, 11 (1984), 1134–1142. https://doi.org/10.1145/1968.1972

Yakir Vizel, Arie Gurfinkel, and Sharad Malik. 2015a. Fast Interpolating BMC. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9206)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, 641–657. https://doi.org/10.1007/978-3-319-21690-4_43

Yakir Vizel, Alexander Nadel, and Vadim Ryvchin. 2015b. Efficient generation of small interpolants in CNF. *Formal Methods Syst. Des.* 47, 1 (2015), 51–74. https://doi.org/10.1007/s10703-015-0224-5

Douglas H Wiedemann. 1987. *Hamming geometry*. Ph.D. Dissertation. University of Waterloo.