Raymond and Beverly Sackler Faculty of Exact Sciences
The Blavatnik School of Computer Science

Towards a Theory of Learning Inductive Invariants

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

by

Yotam M. Y. Feldman

under the supervision of
Professor **Mooly Sagiv** and Professor **Sharon Shoham-Buchbinder**

Submitted to the Senate of Tel Aviv University
August 2022

Abstract

Towards a Theory of Learning Inductive Invariants

Yotam M. Y. Feldman
Doctor of Philosophy
School of Computer Science
Tel Aviv University

SAT-based invariant inference is a prominent approach to automatic safety verification, with algorithms that are successful in practice, but not well-understood. In this thesis we investigate conceptual questions about SAT-based invariant inference algorithms through a theoretical study of their complexity, inspired and influenced by learning theory. The results shed light on the design choices and principles underlying modern SAT-based invariant inference algorithms, and situate invariant inference as a new domain of interest for learning theory.

We develop query-based models that are suitable for learning inductive invariants that can model algorithms of interest, compare the hardness of the query-based invariant inference to classical exact concept learning with queries, prove information-based lower bounds, and study the power of models with rich queries compared to previously-studied models. We derive complexity upper bounds for a variant of interpolation-based inference on an expressive class of inductive invariants, and, borrowing ideas from exact concept learning, construct new algorithms with complexity results for more appealing syntactic forms of invariants, including decision trees. We utilize Bshouty's monotone theory from exact concept learning to cast property-directed reachability as a form of abstract interpretation in a new domain, thereby illuminating how over-approximation is systematically achieved in this technique.

Acknowledgements

I would like to extend my deep gratitude to all those who were part of this work.

First and foremost, to my advisors, Mooly Sagiv and Sharon Shoham, for supporting my growth during this time—intellectual and otherwise. I was fortunate to have advisors that are not only professional, devoted, and exquisite researchers, but also compassionate and open and involved with students' life. This thesis would not have been possible had they not believed in me more than I did. Sharon's dedication, resoluteness, inventiveness, and warmth are a beacon to all who know her. Mooly's optimism, patience, creativity, and undepletable drive are a force of nature. The greatest compliments I receive nowadays is how there are aspects in which I'm similar to Sharon and Mooly.

To Neil Immerman, for the privilege of observing the process by which immense kindness, curiosity, and brilliance forge connections between humans and between mathematical fields. I always brag about having worked with you when an opportunity presents itself.

To James R. Wilcox, for a most wonderful, fruitful and enjoyable collaboration. I learned a lot, not least that the best possible outcome of a collaboration is friendship.

To my collaborators on papers authored during my PhD that are not included in this thesis: John Cyphert, Constantin Enea, Artem Khyzha, Zachary Kincaid, Adam Morrison, Aleksandar Nanevski, Thomas W. Reps, and Noam Rinetzky.

To Alex Aiken, Aleksandar Nanevski, Oded Padon, and Thomas W. Reps, for their kind and generous hospitality.

To the Israeli Academy of Science and to the European Research Council, for their generous financial support.

To my office-mates: Kalev Alpernas, Neta Elad, Asya Frumkin, Elazar Gershuni, Shelly Grossman, Artem Khyzha, Oded Padon, Hila Peleg, Orr Tamir, and Marcelo Taube.

To Kalev Alpernas and Hila Peleg, for a vigorous instant-messaging group and uncountable emotional support, shrewd critique, and good times.

To my family and friends, who were there for me when I struggled and when I didn't.

And to my parents, who brought me thus far.

Contents

List of Tables

List of Figures

List of Algorithms

List of Jokes

Chapter 1

Introduction

> Science may be described as the art of systematic over-simplification.
>
> — Karl Popper

Automatic program verification requires of its creators to write algorithms that can reason about programs and mathematically certify their correctness. The need is pressing and unrelenting: as software systems increasingly permeate day-to-day functions, as well as safety-critical operations, confidence in their correctness is a part of life, to our occasional dismay when that confidence is unjustified; verification's goal is to bring about systems that do merit this confidence, by verifying that it holds. However, those who enjoyed a classical computer science education are likely to protest: the objective of automatic verification makes absolutely no sense, because, theoretically, this is a hopeless endeavor; it can be shown that there is no way to decide whether a program is correct that is more efficient than exhaustively checking all program executions, which is bad when the possible executions are very many (**PSPACE**-complete), and infinitely bad when their number is infinite (undecidable). Why not do machine learning instead?

However, what is in general impossible may still be possible in many practical cases. Systematic, rigorous, and well-reasoned over-simplification is now a cornerstone of how we automatically verify programs despite the theoretical impossibility. How this over-simplification is performed, and why it is performed this way, is the subject matter of this thesis.

Many verification problems boil down to a form of over-simplification called an *inductive invariant*, used to prove *safety*. When an inductive invariant is discovered by the algorithm, it is still unknown exactly which states the program can reach and which states it cannot, but it is certainly known that the program cannot reach a state it should not. The problem of automatically finding such inductive invariants is called *invariant inference*.

In the last two decades, the spectacular achievements of a new approach to invariant inference, called *SAT-based invariant inference*, took the research community (and industry practitioners) by storm. The theory—understanding the complexity and principles of SAT-based invariant

inference algorithms—fell behind.

This thesis undertakes this challenge by linking invariant inference algorithms to another branch of computer science: learning theory. As previously observed, inductive invariants are not unlike data classifiers [SNA12, SGH+13b], yet differ in significant ways [GLMN14], and this has motivated many works that adapt and apply learning techniques to invariant inference [e.g. GLMN14, NMS+20, END+18, JGST10, GNMR16, SGH+13b, SNA12, SGH+13a, SA16, KPIA20, JS17]. This thesis is the first study of invariant inference and learning that

- addresses fundamental questions of complexity and expressive-power for the problem of invariant learning with different kinds of SAT queries, and
- sheds light on some of the central invariant inference algorithms in existence—interpolation-based inference [McM03] and property-directed reachability [Bra11, EMB11]—by viewing them through the lens of learning, even though these algorithms were not inspired by learning, nor widely-understood as such.

Our contributions break new ground in both invariant inference and learning theory:

- From the perspective of invariant inference, we launch a theoretical investigation into how algorithms handle invariants with complex syntactic forms, take on transition systems with challenging reachability patterns, and which invariants they manage to find.
- From the perspective of learning, we introduce models of learning with queries with complexity results that are inherently different from classical concept learning problems, calling for adaptations of existing techniques as well as new ideas that are appropriate for invariant learning.

This is a story of an over-simplification of invariant inference algorithms, and what it can teach us.

1.1 Background

Let us be more precise about the two key players in this thesis—invariant inference and exact concept learning.

1.1.1 Inductive Invariant Inference

We begin with how programs are modeled as transition systems and how they are proven safe using inductive invariant inference.

Transition systems. A program is modeled as transitioning throughout its execution between *states*, as expressed by a *transition relation* $\delta(\sigma, \sigma')$, which indicates that when the program performs a single step starting from state σ it may arrive at the state σ'. From a set of initial

states *Init*, the executions of the system are (finite or infinite) sequences of states $\sigma_0, \sigma_1, \ldots$ such that $\sigma_0 \in \mathit{Init}$ and $(\sigma_i, \sigma_{i+1}) \in \delta$ for every i.

We focus on the fundamental setting of propositional systems, which also applies to infinite-state systems using predicate abstraction [FQ02, GS97], wherein each state σ is a valuation mapping the n Boolean variables of the vocabulary Σ to *true*/*false*. We represent sets of states using formulas—a formula φ represents the set of states satisfying it, $\{\sigma \mid \sigma \models \varphi\}$. We further represent a transition relation using a formula δ, over two copies of the vocabulary Σ, Σ', where Σ denotes the pre-state of the transition and Σ' the post-state, so that $(\sigma, \sigma') \models \delta$ indicates that the system may transition from the state σ to the state σ'.

Safety verification. Safety verification is about proving that the system never reaches a bad state. (This is in contrast to liveness verification, which is about proving that something good eventually happens.) Given a set of initial states *Init*, a transition relation δ, and a set of bad states *Bad*, the goal is to prove that there is no execution $\sigma_0, \sigma_1, \ldots$ of δ from *Init* such that $\sigma_i \in \mathit{Bad}$ for some i. In pure theoretical computer science terms, safety verification is a graph (un)reachability problem, where the graph is over an exponential number of vertices (valuations to the Boolean variables), and the edges are expressed succinctly (through the transition relation δ).

Inductive invariant inference. The inference of inductive invariants is a fundamental technique in safety verification, and the focus of many works [e.g. McM03, Bra11, EMB11, CC77, SGF13, ABD+15, FB18, DDLM13]. The task is to find a formula I that holds in the initial states of the system (as sets of states, $\mathit{Init} \subseteq I$), excludes all bad states ($I \cap \mathit{Bad} = \emptyset$), and is closed under transitions of the system, namely, if $(\sigma, \sigma') \in \delta$ and $\sigma \in I$ then also $\sigma' \in I$. If the latter property does not hold, there is *counterexample to induction*: a transition $(\sigma, \sigma') \in \delta$ such that $\sigma \in I$ but $\sigma' \notin I$. In propositional systems, an inductive invariant exists iff the system is safe. (In infinite-state systems this is not always the case, due to the restricted expressive power of the languages used for invariants, but inductive invariants are still paramount.) In pure theoretical computer science terms, an inductive invariant forms a vertex cut $(I, \neg I)$ in the reachability graph that disconnects *Init*, *Bad*.

SAT-based invariant inference. Over the years, many algorithmic ideas have been proposed to address the challenge of automatic invariant inference. The advance of SAT-based reasoning in the last decades has led to the development of successful algorithms that use a SAT solver to check the satisfiability and validity of formulas, obtain satisfying models, and use the results in the search for an inductive invariant. Two prominent SAT-based algorithms are interpolation-based invariant inference [McM03] and IC3/PDR [Bra11, EMB11]. These algorithms have led to a significant improvement in the ability to verify realistic hardware systems, and have also been extended and generalized to software systems [e.g. CGS10, ALGC12, KW11, AM13, DA16, BGKL13, HB12, KGC14, BG15, KBI+17, CGMT14].

Understanding SAT-based invariant inference. Successful SAT-based inference algorithms are typically tricky and employ many clever heuristics. This is in line with the inherent asymptotic

complexity of invariant inference, which is hard even with access to a SAT solver [LQ09]. However, the practical success of inference algorithms calls for a more refined complexity analysis, with the objective of understanding the principles on which these algorithms are based.

1.1.2 Exact Concept Learning

Cut to a (seemingly) very different scenario.

Concept learning. In this setting, a *learner*'s goal is to identify a concept—an unknown set—based on partial information. We assume that the concept is specified by an unknown propositional formula ψ (a set of satisfying valuations). Other representations of the concept have also been studied and influenced verification (see §3.7), but our focus in this thesis is on SAT-based verification, where formulas are natural.

Probably the most familiar learning setting is when the partial information available to the learner consists of labeled *examples*: the learner has access to valuations $v_1, v_2, \ldots$ that it knows for each one whether $v_i \models \psi$ or not. In PAC learning [Val84], examples are drawn at random, and the learner tries to find a concept that has small expected classification error. In exact concept learning, in contrast, the learner tries to identify ψ exactly (zero classification error) while using queries it can choose.

Exact learning using queries. In this setting, pioneered by Angluin [Ang87b], the learner iteratively presents queries—questions about ψ—to a *teacher*, and uses the teacher's answers to reproduce ψ. Which questions the learner can ask brings about different learning models. The most studied forms of queries are *equivalence* queries (is θ the unknown formula, and if not, what is a differentiating counterexample, a valuation where θ, ψ disagree?) and *membership* queries (is v a model of the unknown formula?). In all cases the formulas/valuations in the queries are chosen by the learner, based on the teacher's responses to previous queries.[1]

Query complexity. A central question in exact learning theory is *query complexity*: how many queries are required for the learner in the worst case to identify any ψ from a certain class of concepts and with certain queries. For example, how many equivalence and membership queries are required to correctly identify a formula out of the class of all formulas that have a short DNF representation? What if the formulas are all *monotone*, that is, where variables appear only positively? What if only equivalence queries are allowed? Such questions are studied in many works [e.g. Ang87b, Ang90, Gav93, AP95, Bsh95, HPRW96, Bsh96, Bsh97, BCG02, Ang04, HKSS12].

The monotone theory. One particular avenue of work that we will use extensively in this thesis is the monotone theory of Bshouty [Bsh95]. The monotone theory provides an understanding of how formulas can be represented as conjunctions of monotone DNF formulas, and how these can be computed efficiently as a way to perform exact concept learning for classes such as the short almost-monotone DNF formulas, and the formulas that have both short CNF and DNF

[1] Bring exact concept learning to your next party instead of the old "Twenty Questions"!

representations (but are not necessarily monotone). At its core, the monotone theory studies the *monotonization* of a formula φ w.r.t. a state b, $\mathcal{M}_b(\varphi)$, which is an approximation of the formula φ such that each variable appears in $\mathcal{M}_b(\varphi)$ only in one polarity (with or without negation) in a way specified by b.

Exact concept learning with queries is the inspiration for this thesis' invariant learning with queries. We prosper on both the similarities—employing algorithms and techniques from exact learning to the study of invariant learning—and the differences, establishing the unique challenges of invariant learning.

1.2 Research Challenges

We now outline the major research challenges this thesis addresses.

1.2.1 Invariant Learning and its Relation to Concept Learning

Approaching the goal of understanding invariant inference by thinking about it as a learning problem first turns the spotlight on how inferring invariants and learning concepts compare. The questions asked, and the answers, clarify the connections between inference and learning, and highlight the unique characters of invariant inference, which we also scrutinize using ideas and tools from learning theory. The apparent similarities and perplexing differences are outlines in the challenges here.

The target of both invariant learning and concept learning is a separator set between good and bad states. The set could be complex to represent in a formula, depending on how the states are "aligned" in the state space. Both invariant inference and concept learning need to address this and efficiently learn a representation of the desired. However, invariant inference and concept learning are different in that in invariant inference, the algorithm does not probe the invariant directly, but rather explores the transition relation, which reflects on the invariant only indirectly. For instance, if a state is neither reachable from the initial states in a small number of steps, nor reaches a bad state in a small number of steps, then in general it is unclear whether that state should be included in the invariant or not. Indeed, this point was the motivation of Garg et al. [GLMN14] for defining the *ICE learning* model, the first learning model tailored to invariant inference.

The comparison between invariant inference and concept learning suggests that it may be fruitful to view invariant inference as a learning problem and that adaptations are necessary [GLMN14]. This raises several questions:

1. **Learning models**: What theoretical models would allow to phrase invariant inference as a learning problem, and existing invariant inference algorithms as learning algorithms? (As we will show, ICE learning cannot model invariant inference algorithms such as interpolation-based inference and PDR.)

2. **Fundamental limits**: What is the complexity of learning invariants? Is the complexity of learning an invariant I the same as the complexity of learning I as a concept?

3. **Using concept learning algorithms**: Can the powerful learning algorithms developed over time be used to create new, powerful invariant inference algorithms? Do existing results in concept learning in fact explain invariant inference algorithms that were developed without attentiveness to techniques in concept learning?

1.2.2 Understanding Invariant Inference Algorithms

The invariant learning viewpoint allows us to tackle some of the most fundamental questions about existing, powerful invariant inference approaches. These include:

1. **Which invariants**: In the space of possible invariants, do algorithms tend to converge to some inductive invariants rather than others? How can algorithms be biased towards a short invariant, and not be led astray towards other, longer invariants?

2. **Syntactic structure**: What syntactic features of the invariant enable the algorithm to find it efficiently, through its short representation, rather than finding the same invariant but through a long equivalent representation?

3. **Overapproximation**: How and when do inference algorithms refrain from the pitfall of computing the set of states reachable in a bounded number of steps and simply increasing the bound? This simple approach is not without merit, but fails in the likely case that some states in the invariant are reachable in only a large number of steps (or not at all).

4. **SAT queries**: What are the requirements from the SAT solver for efficiently implementing algorithms' mitigations for the challenges of invariant inference?

We study these questions for interpolation-based invariant inference in Chapter 4, and for property-directed reachability in Chapter 5, with Chapter 3 studying the the power of rich SAT queries as well as the fundamental complexity limits.

1.3 Contributions

The results in this thesis were published in [FISS20, FSSW21, FSSW22, FS22].[2]

1.3.1 Learning Models and Lower Bounds for Invariant Inference (Chapter 3)

This thesis starts with the investigation of the problem of invariant inference from a complexity-theoretic perspective of learning with queries, its fundamental limits and expressive power with different queries used by invariant inference algorithms.

[2]Not included, yet much cherished, are the results of [FWSS19], as well as [FEM+18, FKE+20].

Query-Based Learning Models for Invariant Inference

Problem formulation: inference of polynomial-length CNF. We introduce the study of *polynomial-length inference*, in which the algorithm is provided with $(Init, \delta, Bad)$, and is required to decide whether there exists an inductive invariant I for the given system, with the additional restriction that I is expressible in a formula of polynomial length in conjunctive normal form (CNF, e.g. $(p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_3 \vee \neg p_4)$). All our results apply equally to inferring invariants in disjunctive normal form (DNF, e.g. $(p_1 \wedge p_2 \wedge p_3) \vee (\neg p_1 \wedge p_3 \wedge \neg p_4)$). (We prefer one form over the other depending on which is natural for the algorithm in consideration.)

The motivation for this definition is that inference algorithms can only succeed when the invariant they infer is not too long. We are most interested in how efficient the search is when it is not "hopeless", when such a short inductive invariant actually exists. The class of polynomial CNF invariants is richer than usually considered in template-based invariant inference [e.g. JSS14, CSS03, SSM04, SG09, SGF13, ABD+15], in line with how recent invariant inference algorithms employ richer syntactical forms of invariants [McM03, Bra11], the motivation being generality of the verification method and potential improvement of the success rate.

We provide a computational hardness result for this problem, showing that polynomial-length invariant inference is complete for Σ_2^P, the second level of the polynomial-hierarchy. This means that the problem is **NP**-hard even with access to a SAT solver. (Without the polynomial-length restriction, it is much harder, **PSPACE**-complete [LQ09].) This strengthens the hardness result of template-based abstraction by Lahiri and Qadeer [LQ09].

Having established computational lower bounds, we move on to analyze the hardness of the problem from a different perspective, that distills the role of how algorithms make progress through answers to SAT queries. We do this by studying query-based learning models and information-based lower bounds.

The Hoare-query model. We introduce a query-based learning model for invariant inference called the *Hoare-query model*. The idea is that the algorithm attempts to learn an inductive invariant while not given direct access to the most interesting part of the input to the problem, the transition relation δ. (It can access $Init, Bad$ freely.) Instead, the algorithm accumulates information of δ by performing a series of *queries*. In the Hoare-query model, algorithms repeatedly choose α, β and query for the validity of the Hoare triple $\{\alpha\}\delta\{\beta\}$, which is valid iff every transition $(\sigma, \sigma') \in \delta$ where $\sigma \in \alpha$ is guaranteed to have $\sigma' \in \beta$. An oracle $\mathcal{H}(\delta, \alpha, \beta)$ answers the queries, and in practice this is implemented using a SAT solver. This is a significant point—the need to define special learning models for invariant inference is that it seems that the classical queries of exact concept learning cannot be answered about the invariant in practice without a-priori knowledge of the invariant [GLMN14]. In contrast, Hoare queries do not require such foresight, and are readily implemented.

The Hoare-query model is general enough to capture algorithms such as PDR and many of its variants, and leaves room for other interesting design choices. For example, PDR maintains a sequence of formulas (frames) $\mathcal{F}_0^{\text{pdr}}, \mathcal{F}_1^{\text{pdr}}, \ldots$. When the algorithm executes, it checks if a frame

$\mathcal{F}_i^{\text{pdr}}$ is inductive, which can be implemented by a Hoare query with $\alpha = \mathcal{F}_i^{\text{pdr}}, \beta = \mathcal{F}_i^{\text{pdr}}$; and it checks whether a generalization clause c is inductive relative to the previous frame $\mathcal{F}_i^{\text{pdr}}$, which amounts to the query $\alpha = \mathcal{F}_i^{\text{pdr}}, \beta = c$. As we will see, it is of utmost importance that in the last query, $\alpha \neq \beta$.

In contrast, the Hoare-query model does not capture white-box approaches, such as any program analyses that operate at the level of the code, such as the sort of abstract interpretation that we are most used to, generating abstract transformers for each instruction in the program's code; but it does capture forms of abstract interpretation such as symbolic abstraction [RSY04, TLLR15].

Queries to capture interpolation-based algorithms. We extend the Hoare-query model to incorporate bounded model checking operations, which are useful for algorithms based on interpolation. In this model, algorithms repeatedly choose α, β and a bound k, and query for whether every state that δ can reach in at most k steps starting from α necessarily satisfies β. The Hoare-query model is obtained by $k = 1$. In order not to grant the queries unreasonable power, k is restricted to be polynomially-bounded by the complexity parameters of the problem. This model captures a model-based interpolation algorithm that we study extensively in Chapter 4.

Alternatively, the original interpolation-based algorithm by McMillan [McM03] uses a procedure that relies on the internals of the SAT solver, without theoretical guarantees on the resulting form of the resulting interpolant. To capture this we extend the model with a query in which it is the responsibility of the oracle to generate an interpolant (but this interpolant could be of exponential CNF size). (This is potentially stronger than the extended Hoare-query model when the model-based interpolation algorithm cannot produce an interpolant in a polynomial number of extended-Hoare queries.)

Complexity of Hoare-query algorithms. The complexity of an invariant inference algorithm in our models is measured by the number of queries it needs to perform in the worst-case to decide whether an invariant $I \in \mathcal{C}$ exists for the given transition system, where $\mathcal{C}$ is a (syntactic) class of formulas.[3] In polynomial-length inference as defined above, $\mathcal{C}$ is the class of CNF formulas of polynomial length.

This complexity measure is information-theoretic, disregarding the computational effort the algorithm spends to compute the queries, which has several benefits for analysis and proving unconditional lower bounds. When proving upper bounds, we consider in addition to the number of queries also the number of time steps the algorithm performs in between the queries.

[3] This might seem different from the identification problem in concept learning, *finding* $I \in \mathcal{C}$, provided that one exists; the decision formulation is easier—because it is always possible to execute an identification algorithm under the assumption that an invariant exists and verify the result—thereby making the lower bounds more significant. Our upper bounds are all based on algorithms that do find an invariant when one exists.

Lower Bound for the Hoare-Query Model

Lower bound. We establish an exponential lower bound on the number of queries required for an algorithm in the Hoare-query model to solve polynomial-length inference. This lower bound is information-based, and holds regardless of how the algorithm computes the next query. We extend the lower bound to the other query-based models for invariant inference discussed above, including the interpolation-query model. This result is expected, but technically interesting and non-trivial, and forms the foundation for our separation results, which are discussed later.

All generalizations are wrong. Conceptually, this impossibility result shows that the problem of generalization from counterexamples in invariant inference is prominently a problem of information; there is no "correct" way to generalize from a counterexample without deeper understanding of the transition relation. In our theoretical learning setting, this information is obtained solely through queries, giving rise to the lower bound; in practice, more information is available, through the syntactic representation of δ, but usually not much is made of it, which points to an interesting direction for future work on SAT-based invariant inference.

Hoare Queries Are Powerful for Learning Invariants: Separation From the ICE Model

When explaining how PDR can be implemented as in the Hoare-query model, we highlighted the fact that PDR uses queries $\mathcal{H}(\delta, \alpha, \beta)$ where $\alpha \neq \beta$. Is this an important characteristic of PDR's generalization procedure? If SAT solvers were not able to answer such queries, would PDR be impossible? Bradley's original explanation, that the basis of PDR is its incremental nature [Bra11, Bra12], certainly asserts that this is important. For the first time, we can justify this intuition theoretically.

From a learning perspective, it is interesting whether general Hoare queries, with $\alpha \neq \beta$, provide additional power to the model. Can some inference problems be solved with a polynomial number of general Hoare queries, or can the same always be achieved using more restricted queries, $\mathcal{H}(\delta, \alpha, \alpha)$? The latter form of queries is captured by the *inductiveness-query* model, in which algorithms repeatedly choose α and query for the validity of the Hoare triple $\{\alpha\}\delta\{\alpha\}$; if it is not valid, the oracle returns a *counterexample to induction*, which is a pair of states $(\sigma, \sigma') \in \delta$ such that $\sigma \models \alpha$ but $\sigma' \not\models \alpha$. The inductiveness-query model is a reformulation of Garg et al.'s ICE learning model [GLMN14], in which the algorithm presents a series of candidate inductive invariants, and learns from the counterexamples. (Our emphasis in our formulation is on how the algorithm chooses candidates, whereas Garg et al. focused on how the algorithm can efficiently find some candidate that is consistent with all the examples.) An inductiveness query is a special case of a Hoare query (a linear number of Hoare queries can also generate a counterexample). But is the Hoare-query model stronger than the inductiveness-query model?

We prove that it is. We construct a class of transition systems and invariants for which there is an efficient algorithm in the Hoare-query model, but prove that *every* algorithm that only

uses inductiveness queries requires an exponential number to find an inductive invariant in the worst case.

The Hoare-query algorithm that achieves the upper bound is PDR with just one frame; this proves that general Hoare queries are inherent to PDR, which cannot be implemented as an algorithm in the ICE model. Indeed, in order to capture PDR, Vizel et al. [VGSM17] extended the ICE model with additional checks, and our result proves that an extension is indeed necessary. (This of course does not prove that PDR is *always* better than *any* ICE algorithm; the exponential complexity of the problem in the general case largely excludes the possibility of such a "silver-bullet" algorithm.)

Invariant Learning Cannot Be Reduced to Concept Learning

We introduced the Hoare-query model because such queries can be implemented in practice, whereas equivalence and membership queries from concept learning cannot. This is somewhat disheartening; it would have been much nicer to apply an algorithm for learning a class of formulas $\mathcal{C}$ to the problem of inferring invariants from $\mathcal{C}$. Can such queries and algorithms nonetheless be implemented on top of the Hoare-query model (which, as we have seen, is very rich)?

We prove that in general the answer is no. We show that neither equivalence nor membership queries can be simulated by a polynomial number of Hoare queries. The proof considers a class $\mathcal{C}$ for which (1) there are efficient concept learning algorithms that use a polynomial (or subexponential) number of equivalence queries, or a polynomial number of both equivalence and membership queries, to learn formulas from $\mathcal{C}$, whereas (2) an exponential lower bound for Hoare-query model exists. The class $\mathcal{C}$ is the set of short CNF formulas that are antimonotone, namely, where variables appear only negated. The upper bounds in exact concept learning are due to Angluin [Ang87b], and the lower bound for the Hoare-query model is the same lower bound discussed above, with the extra observation that the proof uses transition systems that use only monotone invariants.

These results stem from the challenge of complex reachability of the transition system, which is unique to invariant inference. Indeed, our result separating the Hoare-query model from the ICE model, which involves an upper bound on (a very simple) invariant inference algorithm, depends on "neutralizing" reachability-related complications by considering the very special case where every reachable state is reachable in only one step. The topic of how invariant inference algorithms can cope with more complex reachability is the subject of Chapters 4 and 5.

Counterexamples in Invariant Learning Are Inherently Ambiguous

Another conclusion that we can draw from the impossibility of implementing concept-learning queries focuses on the forms of counterexamples that these use, compared to invariant inference. Classically, learning uses examples labelled positive or negative, and seeks to find the correct classifier that distinguishes them. In exact learning, this manifests in that when the learner

receives the differentiating counterexample σ, the learner knows whether σ is a model of the unknown concept (by checking whether σ is a model of the proposed candidate, and negating the answer). In contrast, the natural counterexample to a candidate inductive invariant, a counterexample to induction, is a *pair* of states, (σ, σ'), such that σ is a negative example *or* σ' is a positive example, and the learner cannot tell which is the case. Indeed, this is Garg et al.'s motivation for ICE Learning [GLMN14]—it is unclear how to implement the teacher if it is to return just one example. Our results prove that indeed there is no way to efficiently implement such a teacher using Hoare queries.

This result can also be cast in terms of pure learning: counterexamples to induction are harder for concept learning. For a moment, let us forget all about verification, and focus on the problem of learning, specifically learning from equivalence queries. Recall that in this setting, there is an unknown formula ψ, the learner iteratively proposes candidate formulas θ, and the teacher responds with a counterexample until the learner succeeds in producing $\theta \equiv \psi$. Now suppose that instead of the differentiating counterexamples studied in the classical learning literature, which always constitute a single example σ such that $\sigma \not\models \psi, \sigma \models \theta$ or $\sigma \models \psi, \sigma \not\models \theta$, the form of counterexamples that the teacher provides are *implication examples* [GLMN14]: pairs (σ, σ') such that $\sigma \not\models \psi, \sigma \models \theta$ or $\sigma' \models \psi, \sigma' \not\models \theta$. Our results imply that this form of counterexamples is weaker, namely, it is harder to learn from such counterexamples: there is a class of formulas for which the number of queries a learner requires when the teacher provides implication examples is asymptotically larger than the number of queries that suffice when the learner provides classical differentiating examples. As implication examples are the concept-learning incarnation of counterexamples to induction, this attests to the inherent ambiguity of a counterexample to induction: it is ever unclear whether the candidate should be strengthened to exclude the pre-state (since the case is $\sigma \not\models \psi, \sigma \models \theta$), or weakened to include the post-state (since the case is $\sigma' \models \psi, \sigma' \not\models \theta$)—unless more information is available, as is the case in our results in Chapter 4.

More generally, this result demonstrates that the convergence rate of learning in Counterexample-Guided Inductive Synthesis [e.g. STB+06, JGST10, JS17] depends on the form of examples.

1.3.2 Upper Bounds for Interpolation-Based Invariant Inference (Chapter 4)

As the above lower bounds demonstrate, to infer inductive invariants, it is not enough to know that an invariant of manageable syntactic form *exists* for the given transition system; algorithms must rely on some additional properties of the transition system to find this invariant. In Chapter 4 we focus on conditions that shed light on how this is achieved in *interpolation-based invariant inference*, the first—and extraordinarily influential—approach to SAT-based invariant inference, which was pioneered by McMillan [McM03].

Essentially, an interpolant generalizes a proof of *bounded* unreachability—that the system cannot reach a bad state in k steps (invoking bounded model checking [BCCZ99])—into (what is hopefully) a proof of *un*bounded unreachability, namely, a part of the inductive invariant.

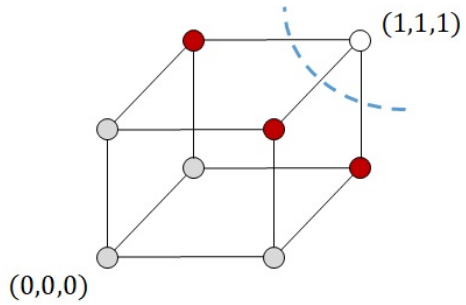


Figure 1.1: The (outer) boundary of an invariant $I = x \wedge y \wedge z$, denoting the singleton set containing the far-top-right vertex of the 3-dimensional Boolean hypercube, $\{(1,1,1)\}$. Its neighbors are I's boundary (depicted in red): $\{(1,1,0),(1,0,1),(0,1,1)\}$. The rest of the vertices are in $\neg I$ but not in the boundary (depicted in gray). (Illustration inspired by [O'D14, Fig. 2.1].)

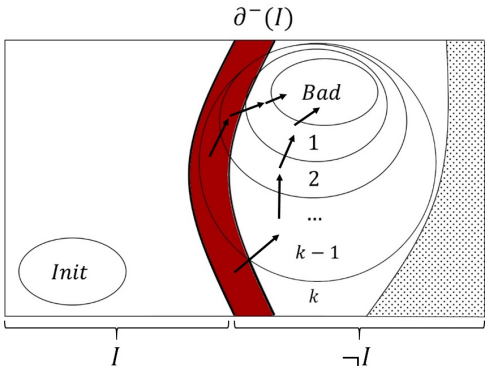


Figure 1.2: An illustration of the fence condition. The boundary $\partial^-(I)$ of the invariant (the states in $\neg I$ nearest to I, in red) are backwards k-reachable (reach a bad state in k steps, for example by the transitions depicted by the arrows), but not all states in $\neg I$ are backwards k-reachable (or even backwards reachable at all, in the dotted area).

The present view is that interpolants strive to abstract away from irrelevant aspects, which is "heuristic in nature" [McM18]. This remark alludes to the fact that each bounded proof may allow several interpolants, which greatly differ from the perspective of invariant inference. Perhaps for this reason, there is currently no theoretical understanding of the efficiency of interpolation-based algorithms. In contrast to this view, we identify conditions that facilitate a theoretical complexity analysis of this algorithmic approach.

We target the line of work of interpolation-based inference that is based on a model-based computation of interpolants [SNA12, CIM12, BGKL13, DA16], whose propositional version was inspired by PDR [CIM12]. We show how a condition on transition systems and invariants, called the *fence* condition, (partially) bridges the gap between exact learning and invariant inference, which allows us to derive efficient upper bounds for the existing algorithm [CIM12, BGKL13] in inferring monotone DNF invariants. This is the first result showing a polynomial upper bound on the complexity with access to a SAT solver as an oracle of an interpolation-based invariant inference algorithm. The same fence condition also allows us to devise new inference algorithms, based on more advanced exact learning algorithms, with efficient complexity results for even larger syntactic classes of invariants.

The Fence Condition

The fence condition relates 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, denoted $\partial^-(I)$, 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. 1.1 for an illustration.) The k-fence condition requires that all the states in I's boundary $\partial^-(I)$ reach a bad state in at most k steps. (See Fig. 1.2.) 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 central insight of the fence condition is that invariant inference behaves well when the reachability of a particular set of states, $\partial^-(I)$, behaves well, and that this set is smaller than the entire $\neg I$.

Upper Bound for the Algorithm By Chockler et al., Bjørner et al. [CIM12, BGKL13]

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 (all variables appear only positively), then the interpolation algorithm by Chockler et al. and Bjørner et al. [CIM12, BGKL13] converges in a linear number of iterations. These results also imply that a *dual* model-based interpolation algorithm can efficiently infer invariants that have a short conjunctive normal form (CNF) that is antimonotone (all variables appear only negatively) under a dual version of the fence condition. This dual algorithm is of special interest, because it is reminiscent of simplifying PDR's frames into accurate bounded model checking (we pick up on the comparison between this and PDR when we study overapproximation in PDR later).

Our upper bound for model-based interpolation is inspired by the upper bound for exact learning monotone DNF formulas by Angluin [Ang87b], and uses a key technical property of monotone formulas that goes back to Quine [Qui54], which was also used in Valiant's famous paper [Val84]. In fact, the mechanics of generalization in the learning algorithms involve the minimization of a term by repeatedly dropping literals (see the exposition by Aizenstein and Pitt [AP95]), which is very similar to how the interpolation-based algorithm operates (as well as PDR). The difference, of course, is the criterion for when a literal can be dropped: in learning, as long as a subset/membership query holds; in interpolation, as long as a bounded model checking query holds. The similarities between the learning and interpolation algorithm prompt us to investigate whether there is a systematic method to convert learning algorithms to interpolation-based invariant inference algorithms, based on the fence condition. As we show next, such a method exists; applied to a learning algorithm for DNF formulas (essentially as formulated in Figure 2 of [AP95]), it produces exactly the inference algorithm by Chockler et al. and Bjørner et al. [CIM12, BGKL13].

Efficient Inference from Exact Learning and the Fence Condition

Trying to implement an exact learning algorithm as an invariant inference algorithm runs into the problem, explained above in §1.3.1, that this is impossible to do—in general. However, when the

fence condition holds, we show that some forms of membership and equivalence queries can be implemented in invariant inference. If the exact learning algorithm is guaranteed to query only on examples that are positive or have Hamming distance one from a positive example, and the fence condition holds, then membership to the invariant can be answered by checking whether a bad state can be reached in k steps (as done in the model-based interpolation algorithm discussed above). The result is an invariant inference algorithm with similar complexity guarantees on the number of queries, albeit different queries—extended Hoare queries. Using this transformation, we obtain an algorithm that efficiently infers inductive invariants beyond the previous result, from the class of *almost-monotone* DNF: formulas where at most $O(1)$ of terms contain negated variables. This is achieved by the transformation of Bshouty's Λ-algorithm for exact learning a formula with a known monotone basis [Bsh95].

Bshouty's paper goes on to perform exact learning to an appealing class of formulas: those that can be expressed using both a short DNF formula *and* as a (different) short CNF formula (CDNF). Unfortunately, this algorithm's queries do not satisfy the requirements for the transformation discussed above. We show a different transformation that can implement any exact learning algorithm from equivalence and membership queries—without restrictions on its queries to query only on specific candidates or examples—at the cost of a stronger condition on the transition system and the invariant, a *double-sided* version of the fence condition: that if $\sigma^+ \in I, \sigma^- \in \neg I$ are of Hamming distance one, then both σ^- can reach a bad state in k steps (this is the fence condition) *and* also σ^+ is reachable from an initial state in k steps. When the invariant satisfies this, every membership query can be implemented (by walking from an initial state to a bad state while performing bounded model checking in both directions). Unfortunately, this condition seems overly restrictive—for example, at most one invariant can satisfy the double-sided fence condition—and so we proceed to invent a specialized algorithm for inferring CDNF invariants.

Efficient Inference of CDNF Invariants

To achieve efficient inference of CDNF invariants that relies on the (one-sided) fence condition, rather than the two-sided fence condition as needed for the translation of Bshouty's learning algorithm to invariant inference, we need to devise a specialized inference algorithm, that is inspired by the CDNF learning algorithm, but differs from it in significant ways. We present an algorithm that is guaranteed to infer an inductive invariant whenever there exists an invariant I that satisfies the fence condition, in a number of queries polynomial in the DNF- and CNF-size of I. In essence, the algorithm is similar to the algorithm by Chockler et al. and Bjørner et al. [CIM12, BGKL13], except that in each iteration our algorithm uses *all* valid minimal terms, instead of just one as in the original algorithm. This makes the candidate invariant grow more quickly, potentially converging faster to an invariant, while relying on the fence condition to ensure that this does not overshoot beyond I. (The idea of taking all generalizations is central in Chapter 5.) We use the ideas from Bshouty's Λ-algorithm [Bsh95] in order to find a

compact representation of the disjunction of all valid terms that result from a counterexample to induction.

The algorithm is inspired by Bshouty's CDNF algorithm, but its structure is significantly different, and itself relies on the fence condition. Indeed, it seems that our algorithm *cannot* be explained as a concept learning algorithm translated to invariant inference; targeting the invariant inference problem directly enables us to prove stronger results about CDNF inference.

The central technical innovation we use in our CDNF inference is an algorithm that performs *super-efficient monotonization*. Bshouty [Bsh95] provided an algorithm to compute the monotonization $\mathcal{M}_b(\varphi)$, but the complexity of this algorithm depends on the DNF size of the original formula φ; this stands in contrast to the DNF size of the output $\mathcal{M}_b(\varphi)$, which is the quantity that controls our algorithm's efficiency, which which may be much smaller (and never larger). To use the super-efficient monotonization procedure for invariant inference, the key idea is that the monotonization of an invariant satisfying the fence condition can be computed through the monotonization of the set of states reachable in at most k steps, and our new monotonization algorithm allows to do this efficiently even when the latter set is complex to represent exactly.

Robustness of the Fence Condition

A significant open problem in formal verification is the *robustness* of a method with respect to revisions in the code. For example, suppose the inference algorithm succeeds 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 model-based 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.

1.3.3 Overapproximation in Property-Directed Reachability From the Monotone Theory (Chapter 5)

The fence condition teaches us that to successfully find an invariant, it is enough that reachability behaves well with respect to only a subset of the states. However, the fence condition does not go far enough, because those states still need to be reachable in a small number of steps. This seems in line with the basic idea of interpolation-based invariant inference: check exact reachability in a small number of steps, then generalize. In Chapter 5 we argue that this is not the case for PDR: this algorithm mitigates the challenge of reachability in what is fundamentally a *non-exact* way. The idea is that in PDR there is latent *abstraction* of reachability in each step; applied successfully this is a form of abstract interpretation. The analogy to the fence

condition is when the reachability of the boundary may involve "jumps", between states that are not actually reachable, yet are connected when viewed through the abstraction.

This is a new way to think about PDR. The theory of abstract interpretation [CC77] provides a rich framework to devise invariant inference algorithms, but the research community views PDR and abstract interpretation as mostly unrelated techniques. In particular, a previous investigation of PDR using the theory of abstract interpretation [RS16] had to employ abstractions that are both far from the usual practice of abstract interpreters, and are also too rich in that they can accommodate many algorithms that are unrelated to PDR (see §5.9). As a result, there is currently no conceptual framework that explains how and when PDR is able to overapproximate beyond exact reachability, which is a key challenge to every invariant inference algorithm. We achieve a surprising result: the monotone theory from exact concept learning [Bsh95] enables viewing PDR as classical abstract interpretation (in a new domain). This draws a deep connection between these techniques, and identifies a form of abstraction performed by PDR that distinguishes it both from explicit enumeration and from other algorithmic approaches.

Λ-PDR: Lower Bounding the Overapproximation of PDR

PDR constructs a sequence of formulas, called *frames*, by blocking counterexamples (states that can reach bad states). Given a counterexample, the algorithm conjoins to the frame a generalization clause that blocks the counterexample and also additional states, but not states reachable in one step from the previous frame (we explain PDR in detail in §5.1.2). Theoretically analyzing the behavior of the algorithm is complicated by its highly nondeterministic nature—it depends on the choices of counterexamples and generalization clauses (many of them affected in practice by idiosyncrasies of the underlying SAT solver), and different choices may lead PDR down radically different paths. To ameliorate this, we present an algorithm, called Λ-*PDR*, which resolves this nondeterminism by using all possible answers to these queries, blocking all counterexamples with all admissible generalizations. The resulting frames are tighter than those of PDR, as they include all lemmas that PDR could learn in any execution (§5.1.3). This provides a theoretical handhold to study PDR: whatever overpproximation we can show is performed in Λ-PDR, is also present in standard PDR's frames.

Overapproximation in Subsequent Frames of Λ-PDR Is Characterized by the Monotone Theory

Λ-PDR uncovers a key aspect of the generalization performed by standard PDR. The frames are usually viewed as a sequence of overapproximations that prove bounded safety with an increasing bound. While correct, this does not capture the full essence of generalization in PDR. In particular, naive exact forward reachability also computes such a sequence, albeit a trivial one. We show that in Λ-PDR—and hence, in PDR—there is an inherent *abstraction* that includes additional states in each frame beyond exact forward reachability.

We show that the relation between successive frames in Λ-PDR is characterized by an operation from Bshouty's monotone theory [Bsh95]. The idea is that taking all the generalizations that block a state b amounts to computing the least b-monotone overapproximation of the post-image of the previous frame (§5.2). The result of this operation always contains the post-image, and can include many more states, depending on the Hamming-geometrical alignment of the post-image and the backward reachable states.

Λ-PDR as Abstract Interpretation

Applied successively, the overapproximation that Λ-PDR performs results in a form of abstract interpretation. To capture this, we introduce a new abstract domain, of the formulas for which backward reachable states form a monotone basis. We show that Λ-PDR can be viewed as computing Kleene iterations with the best abstract transformer in this domain. Standard PDR also operates in the same domain, and its frames overapproximate the Kleene iterations that Λ-PDR performs. This is the first time that the theory of state abstraction is able to explain property-directed generalization (§5.3).

Gaps Between Λ-PDR and Exact Forward Reachability

The abstract interpretation procedure of Λ-PDR leads to exponential gaps from algorithms that do not perform such overapproximation (§5.7). We first show an exponential gap between the number of frames in Λ-PDR and the number of iterations of exact forward reachability, establishing the importance of frames for faster convergence. We then show an exponential gap between the number of frames in Λ-PDR and the unrolling depth in the dual interpolation-based algorithm studied in Chapter 4, showing how the successive overapproximation of Λ-PDR can go beyond the fence condition in tackling the problem of taming the transition system's reachability.

Bounds on the Number of Frames in Λ-PDR

When does the abstract interpretation procedure of Λ-PDR converge in few iterations? We prove an upper bound on the number of frames in Λ-PDR in terms of the DNF size of certain "monotonizations" of the transition relation. Although not always tight, this result sheds light on the benefit of the abstraction in certain cases. The proof brings together results from the monotone theory, abstract interpretation, and diameter bounds for transitions systems. This is done by constructing a (hyper)transition system where the states reachable in i steps correspond to the ith Kleene iteration, and bounding the system's diameter (§5.4–§5.5).

Complexity Results for Λ-PDR

We show complexity results for Λ-PDR by combining our bound on the number of iterations with our super-efficient monotonization algorithm (discussed above in §1.3.2), yielding a bound on the number of SAT calls that is not much worse than the bound on number of iterations.

The key idea is that the DNF size of an abstract iterate is bounded by a quantity related to monotonizations of the transition relation, and our super-efficient monotonization algorithm allows to compute it efficiently w.r.t. the same quantity even though the DNF size of the exact post-image of the previous iterate may be larger.

PDR's Looser Frames Can Be Beneficial

PDR differs from Λ-PDR in that it does not perform the best abstraction in our abstract domain (its frames are still elements of the abstract domain, but potentially less precise). We show that in some cases the abstraction of Λ-PDR is overly precise, whereas the looser frames of standard PDR converge in fewer and smaller frames (§5.8).

1.4 Unifying Threads

A new learning domain, with familiar themes. This thesis casts invariant inference as a form of learning, and many of the results draw on comparisons with exact concept learning. Chapter 3 sets up algorithmic models for invariant learning inspired by concept learning, proves separation results between such models, and further expands on lower bounds showing the fundamental differences between invariant learning and concept learning. The similarities between invariants and concepts bring about the upper bounds of Chapter 4, using concept learning algorithms to understand and derive invariant learning algorithms. Chapter 5 builds extensively on latent similarities between the mechanisms of property-directed reachability and the monotone theory from concept learning.

Principles of SAT-Based invariant inference. Our examination of the principles of invariant inference was provoked by the tantalizing intricacy of PDR. In a sense, this thesis is organized around trying to understand PDR through increasingly sophisticated over-simplifications of PDR:

- In Chapter 3, we study a version of PDR with only one frame that incrementally infers invariants that are completely reachable in one step.
- In Chapter 4, we study a version of PDR that replaces the use of a sequence of frames by bounded model checking (the dual model-based interpolation algorithm we study), with invariants that satisfy the fence condition.
- In Chapter 5, we study a version of PDR that does employ frames for generalization, which goes beyond the fence condition, but sacrifices efficiency due to the use of all possible generalizations.

We hope that further work will eventually lead to efficient complexity results for PDR itself.

As we have seen, this attempt to untangle PDR has led to insights about invariant learning in general.

Chapter 2

Background

This chapter presents background and notation for invariant inference and exact concept learning, both in the realm of propositional logic. This includes background on algorithms—for both problems—that are studied in length in subsequent chapters.

2.1 Propositional Logic

Let Σ be a *propositional vocabulary*, a finite set of Boolean variables $\{p_1, \ldots, p_n\}$. The well-formed formulas, $\text{wff}(\Sigma)$, are defined in the usual way, by structural induction, with the unary connective of negation ($\neg$) and the binary connectives of conjunction ($\wedge$), disjunction ($\vee$), and implication $\rightarrow$. Formally, an atom is well-formed formula, $p_i \in \text{wff}(\Sigma)$; if $\varphi \in \text{wff}(\Sigma)$, then $\neg\varphi \in \text{wff}(\Sigma)$; if $\varphi_1, \varphi_2 \in \text{wff}(\Sigma)$, then $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \varphi_1 \rightarrow \varphi_2 \in \text{wff}(\Sigma)$; the set $\text{wff}(\Sigma)$ is the minimal set of formulas that satisfies these properties.

A *valuation* x is a mapping from Σ to *true* or *false* (or 1/0). We denote the value that x assigns to p by $x[p]$. We write $x[p \mapsto z]$ for the valuation obtained from x by mapping the value z to p and leaving other variables unchanged. The semantics of the satisfaction relation $x \models \varphi$ for a valuation x and a formula $\varphi \in \text{wff}(\Sigma)$ are defined classically. A formula φ is *satisfiable* is there is at least one valuation x s.t. $x \models \varphi$, and is *valid* if $x \models \varphi$ for every valuation x. We write $\varphi \Longrightarrow \psi$ to denote that the formula $\varphi \rightarrow \psi$ is valid.

Literals, Cubes, Clauses, CNF, DNF. A *literal* ℓ is a variable p (positive literal) or its negation $\neg p$ (negative literal). A *clause* is a disjunction of literals. A *term*, or *cube*, is a conjunction of literals; at times, we also refer to it as a set of literals. The *domain* of a cube d, denoted $dom(d)$, is the set of variables that appear in it (positively or negatively). The *cube of a valuation* x, denoted $cube(x)$, is the (full—with domain Σ) cube that is the conjunction of all the literals that are satisfied in x. The only satisfying assignment of $cube(x)$ is x.

A formula is in *disjunctive normal form (DNF)* if it is a disjunction of terms. DNF_m is the set of all DNF formulas with at most m terms. A formula is in *conjunctive normal norm (CNF)* if it is a conjunction of clauses. CNF_m is the set of all CNF formulas with at most m clauses.

2.2 Invariant Inference

2.2.1 States, Transitions Systems, and Inductive Invariants

We consider safety problems defined via formulas in propositional logic. Throughout, fix a propositional vocabulary Σ of n propositional variables ($|\Sigma| = n$).

States. A *state* is a valuation to Σ. The set of all possible states is denoted States[Σ].

Transition systems. A *transition system* is a triple $TS = (Init, \delta, Bad)$ where $Init, Bad \in \text{wff}(\Sigma)$ define the *initial states* and the *bad states*, respectively, and $\delta \in \text{wff}(\Sigma \uplus \Sigma')$ defines the *transition relation*, where $\Sigma' = \{x' \mid x \in \Sigma\}$ is a copy of the vocabulary used to describe the post-state of a transition. A *transition* is a pair of states σ, σ' such that $(\sigma, \sigma') \models \delta$, where in the latter equation the interpretation of Σ is according to σ and Σ' according to σ'. Given a $\varphi \in \text{wff}(\Sigma)$, we denote by φ' the formula obtained from φ by replacing each variable $p \in \Sigma$ with its counterpart $p' \in \Sigma'$. If $\tilde{\Sigma}, \tilde{\Sigma}'$ are distinct copies of Σ, $\delta[\tilde{\Sigma}, \tilde{\Sigma}']$ denotes the substitution in δ of each $p \in \Sigma$ by its corresponding variable in $\tilde{\Sigma}$ and likewise for $\Sigma', \tilde{\Sigma}'$. A *class* of transition systems, denoted $\mathcal{P}$, is a set of transition systems.

Safety. An execution is a sequence $\sigma_0, \sigma_1, \ldots$ (finite or infinite) of states such that $\sigma_0 \models Init$ and $(\sigma_i, \sigma_{i+1}) \models \delta$. A state is *reachable* (from the initial states) if it belongs to some execution of the system. A transition system TS is *safe* if all the states that are reachable from the initial states satisfy $\neg Bad$.

Inductive invariants. An *inductive invariant* for TS is a formula $I \in \text{wff}(\Sigma)$ such that (i) $Init \implies I$, (ii) $I \wedge \delta \implies I'$, and (iii) $I \implies \neg Bad$. In the context of propositional logic, a transition system is safe if and only if it has an inductive invariant.

Counterexamples to induction. When I is not inductive, a *counterexample to induction* is a transition σ, σ' such that $(\sigma, \sigma') \models I \wedge \delta \wedge \neg I'$.

Post-image. Given a set of states $S \subseteq \text{States}[\Sigma]$, the *post-image* is $\delta(S) = \{\sigma' \mid \exists \sigma \in S.\ (\sigma, \sigma') \models \delta\}$. The *reflexive post-image* is $\underline{\delta}(S) = \delta(S) \cup S$. We extend this notation from sets of states to formulas in the natural way, so that $\underline{\delta}(\psi)$ is $\underline{\delta}(S)$ where S is the set of satisfying valuations of ψ, etc.

Bounded reachability and bounded model checking. Given a transition system $(Init, \delta, Bad)$ and a *bound* $k \in \mathbb{N}$, the set of *k-forward reachable states*—the states σ such that there is an execution of δ of length at most k starting from a state in $Init$ and ending at σ—are $\underline{\delta}^k(Init)$. The set of *$k$-backward reachable states*—those that can reach a state in Bad along some execution of length at most k—are $\left(\underline{\delta^{-1}}\right)^k(Bad) = \{\sigma \mid \underline{\delta}^k(\{\sigma\}) \cap Bad \neq \emptyset\}$. *Bounded model checking (BMC)* [BCCZ99] checks whether a set of states described by a formula ψ is

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

$$\text{by checking whether } \psi(\Sigma_0) \wedge \bigwedge_{i=0}^{k-1} \delta(\Sigma_i, \Sigma_{i+1}) \wedge \left(\bigvee_{i=0}^{k} \mathit{Bad}(\Sigma_i)\right) \text{ is unsatisfiable.}$$

($\Sigma_0, \ldots, \Sigma_k$ are distinct copies of the vocabulary Σ.) A k-BMC check is either one, with bound k. With similar notation, the set of (forward-) reachable states as $\underline{\delta}^{\omega}(\mathit{Bad}) = \bigcup_{i=0}^{\infty} \underline{\delta}^{i}(\mathit{Init})$, and the set of backward-reachable states is $\left(\underline{\delta^{-1}}\right)^{\omega}(\mathit{Bad}) = \bigcup_{i=0}^{\infty} \left(\underline{\delta^{-1}}\right)^{i}(\mathit{Bad})$.

Duality of Backward- & Forward- Reachability. Invariant inference algorithms often employ forms of forward- and backward- reachability analysis in sophisticated ways. Their roles in the algorithm can always be reversed, giving rise to the *dual* (or *reverse*) algorithm.[1] This is obtained by executing the original algorithm on a dual of the transition system: The dual transition system is $(\mathit{Init}, \delta, \mathit{Bad})^* \stackrel{\text{def}}{=} (\mathit{Bad}, \delta^{-1}, \mathit{Init})$ where δ^{-1} is the inverse (between pre- and post-states) of the transition relation, obtained from δ by switching the roles of Σ and Σ'. The dual of a class of transition systems is the class of dual transition systems. The dual of a formula is $\varphi^* \stackrel{\text{def}}{=} \neg\varphi$. The dual of a class of formulas is the class of dual formulas. We have that I is an inductive invariant w.r.t. $(\mathit{Init}, \delta, \mathit{Bad})$ iff I^* is an inductive invariant w.r.t. $(\mathit{Init}, \delta, \mathit{Bad})^*$. For an algorithm $\mathcal{A}$, The *dual algorithm* $\mathcal{A}^*$ is the algorithm that, given as input $(\mathit{Init}, \delta, \mathit{Bad})$, executes $\mathcal{A}$ on $(\mathit{Init}, \delta, \mathit{Bad})^*$ and returns the dual invariant. For this thesis, the dual algorithm is useful to translate complexity results from the inference of CNF invariants to the inference of DNF invariants and vice versa (see Lemma 3.2.5).

2.2.2 Invariant Inference Algorithms

This section provides very brief technical introductions to the invariant inference algorithms that we study in depth later in the thesis.

Exact Forward- and Backward-Reachability

Exact forward reachability computes the set of states reachable in i steps, $\mathcal{R}_i = \underline{\delta}^{i}(\mathit{Init})$, with increasing values of i, until convergence $\mathcal{R}_i = \mathcal{R}_{i+1}$, in which case $\mathcal{R}_i$ is the set of all reachable states (in any number of steps), and in particular an inductive invariant (the so-called *least fixed-point*). The dual algorithm, *exact backward reachability*, computes $\mathcal{B}_i = \left(\underline{\delta^{-1}}\right)^{i}(\mathit{Bad})$ with increasing values of i, until convergence $\mathcal{B}_i = \mathcal{B}_{i+1}$, in which case $\mathcal{B}_i$ is the set of states that can reach a bad state (in any number of steps), and $\neg\mathcal{B}_i$ is an inductive invariant (the so-called *greatest fixed-point*).

Another way to express these algorithms—which in itself is not efficient, but inspires more efficient algorithms—is when the computation of the exact post-image (or exact pre-image) is done by enumerating states one by one. In that case, a simpler presentation of these algorithms (blurring somewhat the distinction between intermediate iterations but converging to the same invariants) is as follows: Exact forward reachability operates by starting from the candidate

[1] This is true when the transition relation is expressed as by a formula, as we assume here, and as is standard in software model-checking. When the algorithm operates with a different representation of the transition relation, complications arise; thus for example ternary simulation in PDR [EMB11] uses a netlist repsentation of the hardware circuit, necessitating new ideas for implementing the reverse algorithm [SS17].

$\varphi = \mathit{Init}$, iteratively sampling counterexamples to induction of, $(\sigma, \sigma') \models \varphi \wedge \delta \wedge \varphi'$, and updating the candidate to $\varphi \vee \mathit{cube}(\sigma')$. Likewise, backward forward reachability operates by starting from the candidate $\varphi = \neg \mathit{Bad}$, iteratively sampling counterexamples to induction, $(\sigma, \sigma') \models \varphi \wedge \delta \wedge \varphi'$, and updating the candidate to $\varphi \wedge \neg \mathit{cube}(\sigma')$.

More advanced algorithms are often informally referred to as some mix of forward and backward reachability.

IC3/PDR

Algorithm 1 PDR [Bra11, EMB11]

```
1:  procedure PDR(Init, δ, Bad)
2:      F_0^pdr ← Init
3:      N ← 0
4:      while ∀1 ≤ i ≤ N. F_i^pdr ⟹̸ F_{i-1}^pdr do
5:          F_{N+1}^pdr ← true
6:          while F_{N+1}^pdr ⟹̸ ¬Bad do
7:              for σ_b ∈ F_{N+1}^pdr ∧ Bad do
8:                  BLOCK(σ_b, N + 1)
9:          N ← N + 1
10:     return F_i^pdr such that F_i^pdr ⟹ F_{i-1}^pdr
11: procedure BLOCK(σ_b, i)
12:     if i = 0 then
13:         unsafe
14:     while δ(F_{i-1}^pdr) ⟹̸ ¬σ_b do
15:         take σ s.t. σ ⊨ F_{i-1}^pdr, (σ, σ_b) ⊨ δ
16:         BLOCK(σ, i − 1)
17:     take c minimal s.t. c ⊆ ¬σ_b and δ(F_{i-1}^pdr) ⟹ c
18:                                  and Init ⟹ c
19:     for 1 ≤ j ≤ i do
20:         F_j^pdr ← F_j^pdr ∧ c
```

IC3/PDR [Bra11, EMB11] maintains a sequence of formulas $\mathcal{F}_0^{\text{pdr}}, \mathcal{F}_1^{\text{pdr}}, \ldots$, called *frames*, each of which can be understood as a candidate inductive invariant. The sequence is gradually modified and extended throughout the algorithm's run. It is maintained as an approximate reachability sequence, meaning that (1) $\mathit{Init} \implies \mathcal{F}_0^{\text{pdr}}$, (2) $\mathcal{F}_i^{\text{pdr}} \implies \mathcal{F}_{i+1}^{\text{pdr}}$, (3) $\delta(\mathcal{F}_i^{\text{pdr}}) \implies \mathcal{F}_{i+1}^{\text{pdr}}$, and (4) $\mathcal{F}_i^{\text{pdr}} \implies \neg \mathit{Bad}$. These properties ensure that $\mathcal{F}_i^{\text{pdr}}$ contains the set of states reachable in i steps. (But $\mathcal{F}_i^{\text{pdr}} \implies \neg \mathit{Bad}$ does *not* imply that a bad state is unreachable in *any* number of states.) The algorithm terminates when one of the frames implies its preceding frame ($\mathcal{F}_i^{\text{pdr}} \implies \mathcal{F}_{i-1}^{\text{pdr}}$), in which case it constitutes an inductive invariant, or when a counterexample trace is found. A simple variant of how the frames are constructed appears in Alg. 1, and explained in Chapter 5 (§5.1.2). Essentially, the algorithm produces counterexamples, which are bad states or states that can reach a bad state, and refines the frames to exclude them. Aiming for efficient convergence, PDR chooses to *generalize* and exclude more states with every counterexample. A basic form of generalization starts with $\mathit{cube}(\sigma)$ and drops literals as long as the result is still unreachable from the previous frame. (As we will show, this procedure links exact learning and invariant inference, and will play an important role in all chapters of the thesis.)

For a more complete presentation of PDR and its variants as a set of abstract rules that may be applied nondeterministically see e.g. [HB12, GI15]. The validity of implications is checked using a SAT solver. For example, checking whether $\delta(\mathcal{F}_{i-1}^{\text{pdr}}) \implies c$ in line 17 is implemented by checking the satisfiability of the formula $\mathcal{F}_{i-1}^{\text{pdr}} \wedge \delta \wedge \neg c'$.

Interpolation-Based Inference

Algorithm 2 Interpolation-based invariant inference [McM03]

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

The idea of interpolation-based algorithms, first introduced by McMillan [McM03], is to generalize proofs of bounded unreachability into elements of a proof of *un*bounded reachability, utilizing *Craig interpolation*. Briefly, this works as follows: encode a bounded reachability from a set of states φ in k steps, and use a SAT solver to find that this cannot reach *Bad*. In that case the SAT solver can produce an interpolant χ: a formula representing a set of states that (i) overapproximates the set of states reachable from φ in k_1 steps, and still (ii) cannot reach *Bad* in k_2 steps (any choice $k_1 + k_2 = k$ is possible). Thus χ overapproximates concrete reachability from φ without reaching a bad state, although both these facts are known in only a bounded number of steps. The hope is that χ would be a useful generalization to include as part of the invariant. The original algorithm [McM03], displayed in Alg. 2, sets some k as the current unrolling bound, starts with $\varphi = \mathit{Init}$, obtains an interpolant χ with $k_1 = 1, k_2 = k - 1$, disjoins the interpolant to φ, and continues in this fashion, until an inductive invariant is found, or *Bad* becomes reachable in k steps from φ, in which case k is incremented and the algorithm is restarted. The use of interpolation and generalization from bounded unreachability has been used in many works since [e.g. VG09, McM06, JM07, HJMM04, VGS13]. Combining ideas from interpolation and PDR has also been studied [e.g. VG14]. In Chapter 4 we study in detail a variant of this algorithm that computes interpolants by sampling post-states of the current candidate [CIM12, BGKL13].

ICE

The ICE framework [GLMN14, GNMR16] is a learning framework for inferring invariants from positive, negative and implication counterexamples. (It was later extended to general Constrained-Horn Clauses [END+18].) Here we review the framework using the original terminology and notation; in Chapter 3 (§3.5.1) we use a related formulation that emphasizes the choice of candidates.

In ICE learning, the teacher holds an unknown target (P, N, R), where $P, N \subseteq D, R \subseteq D \times D$ are sets of examples. The learner's goal is to find a hypothesis $H \in C$ s.t. $P \subseteq H, N \cap H = \emptyset$, and for each $(x, y) \in R$, $x \in H \implies y \in H$. The natural way to cast inference in this framework

is, given a transition system $(\mathit{Init}, \delta, \mathit{Bad})$ and a set of candidate invariants $\mathcal{L}$, to take D as the set of program states, P a set of reachable states including Init, N a set of states including Bad from which a safety violation is reachable, R the set of transitions of δ, and $C = \mathcal{L}$. Iterative ICE learning operates in rounds. In each round, the learner is provided with a sample—(E, B, I) s.t. $E \subseteq P, B \subseteq N, I \subseteq R$—and outputs an hypothesis $H \in C$. The teacher returns that the hypothesis is correct, or extends the sample with an example showing that H is incorrect. The importance of implication counterexamples is that they allow implementing a teacher using a SAT/SMT solver without "guessing" what a counterexample to induction indicates [GLMN14, LMN16]. Examples of ICE learning algorithms include Houdini [FL01] and symbolic abstraction [RSY04, TLLR15], as well as designated algorithms [GLMN14, GNMR16].

Theoretically, the analysis of Garg et al. [GLMN14] focuses on *strong convergence* of the learner, namely, that the learner can always reach a correct concept, regardless of how the teacher chooses to extend samples between rounds. We will say that the learner is *strongly-convergent* with *round-complexity* r if for every ICE teacher, the learner finds a correct hypothesis in at most r rounds, provided that one exists. We extend this definition to a class of target descriptions in the natural way.

Abstract Interpretation

A pillar of verification and static analysis, pioneered by Cousot and Cousot [CC77], the central idea of abstract interpretation is to infer inductive invariants by "interpreting" the program over "abstract" semantics induced by the use of an abstract domain. In a nutshell, whereas the reachable states of the system can be accumulated by repeatedly applying the post-image operator, $\mathcal{R}_0 = \mathit{Init}, \mathcal{R}_{i+1} = \underline{\delta}(\mathcal{R}_i)$, until convergence, abstract interpretation works by repeatedly applying an abstract transformer $\xi_{i+1} = \underline{\delta^\sharp}(\xi_i)$ over an algebraic lattice, with the property that the abstract element ξ_0 represents a set of states that includes at least $\mathcal{R}_0$, and $\underline{\delta^\sharp}(\cdot)$ yields an abstract element that includes at least the original, exact post-image $\underline{\delta}(\cdot)$. (Chapter 5 provides a more technical introduction to abstract interpretation (§5.3.1).)

Take, for example, the technique of conjunctive predicate abstraction [FQ02, GS97]. Each abstract element is a subset of predicates, and it represents the set of states that satisfy all the predicates in the subset. We take ξ_0 to be the set of all predicates that hold in every initial state. $\delta^\sharp$ takes an abstract element ξ, and returns the set of predicates that hold in all the states in the post-image of the set of states represented by ξ. Then it can be shown that this procedure converges to the strongest inductive invariant that is expressible in this abstract domain, namely, the conjunction of the predicates that hold in every reachable state of the system.

In predicate abstraction, $\delta^\sharp(\xi)$ is typically computed using a series of SAT queries. Another notable SAT-based abstract interpretation procedure is symbolic abstraction [RSY04, TLLR15]. Abstract interpretation in richer logics has also been widely studied: with arithmetic [e.g. CH78, Min06] and quantifiers [e.g. FQ02, LB07, AHH13, AHH15].

2.3 Exact Concept Learning With Queries

2.3.1 Model and Queries

In *exact concept learning* [Ang87b], the algorithm's task is to identify an unknown formula[2] ψ using queries it poses to a *teacher*. The most studied queries are:

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

The prototypical question studied is how many equivalence and membership queries suffice to correctly identify an unknown ψ from a certain (syntactical) class $\mathcal{L}$.

Another type of query that is less common but will be useful in this thesis is

- *Subset*: The algorithm *chooses* a formula θ, and the teacher answers whether $\theta \Longrightarrow \psi$.

Again, the question is how many queries the algorithm requires.

We are also interested in the total number of computational steps the algorithm takes between queries, to which we refer as the algorithm's running time.

2.3.2 Exact Learning Monotone Formulas

One of the first classes of formulas to be shown efficiently learnable is the class of monotone DNF formulas.

Definition 2.3.1 (Monotone DNF)**.** *A formula* $\psi \in \text{Mon-DNF}_m$ *if it is in DNF with* m *terms, and variables appear only positively.*

For example, $(p_1 \wedge p_3) \vee (p_1 \wedge p_2 \wedge p_4)$ is monotone, while $(\neg p_1 \wedge p_3) \vee (p_1 \wedge p_2 \wedge p_4)$ is not.

The classical property of monotone formulas that make them amenable to efficient learning concerns their *prime implicants*:

Definition 2.3.2 (Prime Implicant)**.** *A term* $\wedge s$*, where* s *is a set of literals, is a* prime implicant *of a formula* ψ *if* $(\wedge s) \Longrightarrow \psi$*, but for every* $\ell \in s$*,* $(\wedge (s \setminus \{\ell\})) \not\Longrightarrow \psi$*. It is* non-trivial *if* $\wedge s \not\equiv$ *false.*

The virtue of monotone formulas is that their prime implicants are all syntactic terms of the formula:

Theorem 2.3.3 ([Qui54])**.** *Let* s *be an non-trivial implicant term of a monotone DNF formula* ψ*. Then there is a term* $\hat{s}$ *of* ψ *such that*

[2] In general, a concept is a set of elements; here we focus on logical concepts.

1. *$\hat{s} \subseteq s$ (as sets of literals).*

2. *If s is prime, then $s = \hat{s}$ (as sets of literals). Namely, s is a syntactic term of ψ.*

Proof. Let v be the valuation that assigns *true* to the literals in ψ and *false* to others; it is well-defined because $p, \neg p$ cannot both appear in s. Then $v \models s$. Since s is an implicant of ψ, also $v \models \psi$. This is a disjunction, so there is some term $\hat{s}$ of ψ such that $v \models \psi$. Since $\hat{s}$ is a conjunction, $v \models \ell$ for every $\ell \in \hat{s}$, and this occurs only when $\hat{s} \subseteq s$.

For the second part, if $s \neq \hat{s}$, then it is impossible for s to be a prime implicant, since dropping any $\ell \in s \setminus \hat{s}$ would also yield an implicant of $\hat{s}$ (since is a conjunction of literals) and thus of ψ (since it is a disjunction of terms). Hence $s = \hat{s}$. □

Unate DNF. A similar property holds for the generalization of monotone formulas to *unate* formulas:

Definition 2.3.4 (Unate)**.** *A formula ψ in DNF/CNF is* unate *if each variable appears with a single polarity, that is, every $p \in \Sigma$ is either always negated or always un-negated in terms/clauses of ψ where p appears.*

Corollary 2.3.5. *Let s be an non-trivial implicant term of a unate DNF formula ψ. Then there is a term $\hat{s}$ of ψ such that*

1. *$\hat{s} \subseteq s$ (as sets of literals).*

2. *If s is prime, then $s = \hat{s}$ (as sets of literals). Namely, s is a syntactic term of ψ.*

Proof. Consider a variable renaming η that maps each $p \in \Sigma$ either to p or to $\neg p$. Extend η to formulas in the natural way (by structural induction). Then $\eta(s)$ is a (prime) implicant of $\eta(\psi)$ for every such translation. Choose a translation η that makes $\eta(\psi)$ monotone. Let s' be a term as guaranteed from Thm. 2.3.3 applied to $\eta(\psi), \eta(s)$. Choose $\hat{s}$ such that $\eta(\hat{s}) = s'$. □

Results about monotone formulas can sometimes be extended to unate ones, and we indicate where this can be achieved through essentially the same arguments.

Algorithms

Angluin [Ang87b] proved that there is an algorithm for learning monotone DNF formulas that uses a polynomial number of equivalence and membership queries. For the purpose of Chapter 4, we first show a somewhat more intuitive algorithm that also achieves efficient learning of monotone DNF, albeit uses stronger subset queries instead of membership (and in addition to equivalence queries); this algorithm's benefit is that it is appropriate for learning any DNF formula, although it is provably efficient only when the formula is monotone. Our presentation is based on Figure 2[3] in Aizenstein and Pitt [AP95] In the code, SubsetQuery (θ)

Algorithm 3 DNF learning [Val84, Ang87b, AP95]

```
1:  procedure LEARN-DNF
2:      φ ← false
3:      while σ′ ← EquivalenceQuery(φ) is not ⊥ do
4:          d ← cube(σ′)
5:          for ℓ in d do
6:              // (intentionally left blank)
7:              // (intentionally left blank)
8:              if SubsetQuery(d \ {ℓ}) = true then
9:                  d ← d \ {ℓ}
10:         φ ← φ ∨ d
11:     return φ
```

Algorithm 4 Monotone DNF learning [Ang87b]

```
1:  procedure LEARN-MONOTONE-DNF
2:      φ ← false
3:      while σ′ ← EquivalenceQuery(φ) is not ⊥ do
4:          d ← ∧{p_i | σ′[p_i] = true}
5:          for ℓ in d do
6:              let x be the state s.t.
7:                  x[p_i] = true iff p_i ∈ d \ {ℓ}
8:              if MembershipQuery(x) = true then
9:                  d ← d \ {ℓ}
10:         φ ← φ ∨ d
11:     return φ
```

returns *true* iff $\theta \Longrightarrow \psi$ where ψ the unknown formula. EquivalenceQuery (H) returns $\bot$ if $H \equiv \psi$ and a differentiating counterexample otherwise.

Theorem 2.3.6. *Alg. 3 successfully learns every ψ, and if $\psi \in$ Mon-DNF$_m$ then it does so with at most $m+1$ equivalence queries, mn subset queries, and $O(nm)$ time.*

Proof. First, always $\varphi \Longrightarrow \psi$, by induction on iterations of the loop in line 4; initially this holds trivially, and continues to hold because the algorithm always adds to φ a disjunct that is either σ', which is a positive example ($\sigma' \models \psi$) by the induction hypothesis, or some set obtained from it that is still contained in ψ thanks to the subset query. φ strictly increases in each iteration, so it must converge to ψ after finitely many steps.

Assume now that $\psi \in$ Mon-DNF$_m$. Then in each iteration, by construction (in line 8), the added to φ is a prime implicant of ψ. By Thm. 2.3.3 this is a syntactic term of ψ, of which there are at most m. Each iteration uses one equivalence query and n subset queries; the last, successful equivalence query adds another one. □

The algorithm by Angluin [Ang87b] replaces the subset query by a clever membership query. A similar pattern appears in Bshouty's algorithms in the monotone theory, discussed in §2.4, so it is worth discussing it here as a preview. (The algorithm is presented in Alg. 4; in the code, MembershipQuery (x) returns *true* iff $x \models \psi$ where ψ the unknown formula.)

In Angluin's algorithm, each iteration produces a term that is monotone by construction, by dropping all the negative literals in $cube(\sigma')$ in line 8, resulting in a monotone initial d. Then, to check whether another literal can be dropped while satisfying the subset query of line 8, the algorithm constructs an example x where $x[p] = true$ iff $p \in d \setminus \{\ell\}$; because ψ is assumed to be monotone, indeed $x \models \psi$ iff $d \setminus \{\ell\} \subseteq \psi$. In this way the algorithm can generate a prime implicant of ψ through standard membership queries rather than subset queries. Bshouty's work employs a similar procedure but with a slightly different objective and justification—see Lemma 2.4.7.

[3] Replacing random examples by examples from equivalence queries.

Dual notions for CNF

Definition 2.3.7 (Antimonotone CNF)**.** *A formula $\psi \in$ Mon-CNF$_m$ if it is in CNF with m clauses, and variables appear only negatively.*

Definition 2.3.8 (Prime Consequence)**.** *A clause c is a* consequence *of ψ if $\psi \implies c$. A* prime consequence*, c, of ψ is a minimal consequence of ψ, i.e., no proper subset of c is a consequence of ψ.*

Theorem 2.3.9. *Let c be an non-trivial consequence clause of an antimonotone CNF formula ψ. Then there is a clause $\hat{c}$ of ψ such that*

1. *$\hat{c} \subseteq s$ (as sets of literals).*
2. *If c is prime, $c = \hat{c}$ (as sets of literals). Namely, c is a syntactic clause of ψ.*

Proof. c is a consequence clause of ψ iff $\neg c$ is a prime implicant of $\neg\psi$. Apply Thm. 2.3.3. □

Corollary 2.3.10. *Let c be an non-trivial consequence clause of unate CNF formula ψ. Then there is a clause $\hat{c}$ of ψ such that*

1. *$\hat{c} \subseteq s$ (as sets of literals).*
2. *If c is prime, $c = \hat{c}$ (as sets of literals). Namely, c is a syntactic clause of ψ.*

Proof. c is a consequence clause of ψ iff $\neg c$ is a prime implicant of $\neg\psi$. Apply Corollary 2.3.5. □

Algorithms for exact learning antimonotone CNF can be obtained by learning the negation of the formula, which is a monotone DNF.

2.4 The Monotone Theory

In this section, we present the monotone theory by Bshouty [Bsh95]. In exact learning, the overarching motivation for the monotone theory is to facilitate learning of formulas beyond monotone. Clearly, a DNF formula where each variable appears only once can be made monotone by a translation, replacing some variables with their negations, but this cannot be done in general. In learning based on the monotone theory, a formula is derived from a conjunction of such formulas, each can become monotone by a *different* translation.

§2.4.1 considers what happens when we "monotonize" a formula to use each variable in one, specified polarity; §2.4.2 studies the conjunction of several such monotonizations, through the *monotone hull* operator; and §2.4.3 discusses when such a conjunction reconstructs the original formula. §2.4.4 outlines how these constructs are used for exact learning.

In this thesis, we employ the monotone theory in Chapter 4 to infer invariants beyond monotone formulas, and in Chapter 5 to analyze overapproximation in PDR. The latter has quite a different flavor from the original application of the monotone theory, yet uses the

same technical material. In light of this, our presentation (which differs somewhat from the original) emphasizes two complementary viewpoints of the monotone's theory main elements: the syntactic view, useful for Chapter 4, where the monotonization is perceived as deleting literals with the wrong polarity; and the semantic-geometric view, useful for Chapter 5, where monotonization is a form of closure operator originating from visibility-like [e.g. O'R04] questions in the Hamming cube.

2.4.1 Least b-Monotone Overapproximations

Our definitions and claims concerning b-monotone overapproximations generalize Bshouty [Bsh95] by considering a partial cube b, and coincide with the original in the case of a full cube. Let us first define the relevant concepts, and then discuss intuitions.

Definition 2.4.1 (b-Monotone Order [Bsh95])**.** *Let b be a cube. We define a partial order over states where $v \leq_b x$ when x, v agree on all variables not present in b, and x disagrees with b on all variables on which also v disagrees with b: $\forall p \in \Sigma.\ x[p] \neq v[p]$ implies $p \in dom(b) \wedge v[p] = b[p]$.*

Intuitively, $v \leq_b x$ when x can be obtained from v by flipping bits to the opposite of their value in b.

Definition 2.4.2 (b-Monotonicity [Bsh95])**.** *A formula ψ is b-monotone for a cube b if $\forall v \leq_b x.\ v \models \psi$ implies $x \models \psi$.*

That is, if v satisfies ψ, so do all the states that are farther away from b than v. For example, if ψ is 000-monotone and $100 \models \psi$, then because $100 \leq_{000} 111$ (starting in 100 and moving away from 000 can reach 111), also $111 \models \psi$. In contrast, $100 \not\leq_{000} 011$ (the same process cannot flip the 1 bit that already disagrees with 000), so 011 does not necessarily belong to ψ.

Definition 2.4.3 (Least b-Monotone Overapproximation [Bsh95])**.** *Given a formula φ and a cube b, the* least b-monotone overapproximation *of φ is a formula $\mathcal{M}_b(\varphi)$ defined by*

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

For example, if $100 \models \varphi$, then $100 \models \mathcal{M}_{000}(\varphi)$ because $\mathcal{M}_{000}(\varphi)$ is an overapproximation, and hence $111 \models \mathcal{M}_{000}(\varphi)$ because it is 000-monotone, as above. Here, thanks to minimality, 011 does not belong to $\mathcal{M}_{000}(\varphi)$, unless 000, 001, 010, or 011 belong to φ.

The minimality property of $\mathcal{M}_b(\varphi)$ is formalized as follows:

Lemma 2.4.4. *$\mathcal{M}_b(\varphi)$ (Def. 2.4.3) is the least b-monotone formula ψ (Def. 2.4.2) s.t. $\varphi \Longrightarrow \psi$ (i.e., for every other b-monotone formula ψ, if $\varphi \Longrightarrow \psi$ then $\mathcal{M}_b(\varphi) \Longrightarrow \psi$).*

Proof. That $\mathcal{M}_b(\varphi)$ is b-monotone overapproximation of φ is immediate from the definition. For minimality, let ψ be a b-monotone formula s.t. $\varphi \Longrightarrow \psi$, we need to show that $\mathcal{M}_b(\varphi) \Longrightarrow \psi$. Let $x \models \mathcal{M}_b(\varphi)$. Then, by definition, there is $v \models \varphi$ s.t. $v \leq_b x$. By the assumption that $\varphi \Longrightarrow \psi$

also $v \models \psi$, and, because ψ is assumed to be b-monotone, it follows that $x \models \psi$. The claim follows. □

$\mathcal{M}_b(\varphi)$ is a well-defined overapproximation of φ. Its main significance for learning theory is that it can be computed efficiently (through the DNF representation we also show below), obtaining the original φ as the conjunction of least b-monotone overapproximations (with different b's).

A technical observation that will prove useful several times is that $\mathcal{M}_b(\cdot)$ is a monotone operator:

Lemma 2.4.5. *If $\varphi_1 \Longrightarrow \varphi_2$ then $\mathcal{M}_b(\varphi_1) \Longrightarrow \mathcal{M}_b(\varphi_2)$.*

Proof. Immediate from the definition of $\mathcal{M}_b(\cdot)$. □

Geometric intuition. Geometrically, ψ is b-monotone if $v \models \psi \Longrightarrow x \models \psi$ for every states v, x s.t. $v \leq_b x$; the partial order $\leq_b$ indicates that x is "farther away" from b in the Hamming cube than v from b, namely, that there is a shortest path w.r.t. Hamming distance from b to x (or from $\pi_b(x)$—the projection of x onto b—to x, when b is not a full cube) that goes through v. A formula ψ is b-monotone when it is closed under this operation, of getting farther from b. In this way, $\mathcal{M}_b(\varphi)$ corresponds to the set of states x to which there is a shortest path from b that intersects φ.[4]

Syntactic intuition. Ordinary monotone formulas are $\overline{0}$-monotone; for general b, a formula ψ in DNF is b-monotone if interchanging $p, \neg p$ whenever $b[p] = \mathit{true}$ results in a formula that is monotone DNF per the standard definition.[5] This is formalized by the following lemma.

Lemma 2.4.6. *Suppose that ψ is expressible in DNF such that every variable $p \in \mathit{dom}(b)$ appears in ψ in polarity according to $\neg b[p]$: p appears only positively if $b[p] = \mathit{false}$ and only negatively if $b[p] = \mathit{true}$. (Variables $p \notin \mathit{dom}(b)$ are unconstrained, and can appear both positively and negatively in different terms of the formula.) Then ψ is b-monotone.*

Proof. Assume $v \leq_b x, v \models \psi$, let t be an arbitrary term of the DNF representation of ψ from the premise; it suffices to prove that $x \models t$. Every literal $\ell \in t$ is satisfied by $v \models \ell$, and hence includes the variable p in the polarity $v[p]$. If $p \notin \mathit{dom}(b)$, the definition of $v \leq_b x$ requires that $v[p] = x[p]$, and so also $x \models \ell$. Otherwise, the premise regarding the polarity of variables in ψ implies that $v[p] = \neg b[p]$; in this case, the definition of $v \leq_b x$ requires that also $x[p] = \neg b[p]$, in which case again $v[p] = x[p]$ and so also $x \models \ell$. The claim follows. □

[4] This is reminiscent of visibility in Euclidean geometry [e.g. O'R04]: picturing b as a guard, the source of visibility, then $\mathcal{M}_b(\varphi)$ is the set of states that are visible in $\neg\varphi$, that is, the set of states σ s.t. the "line segment" $[b, \sigma]$ is contained in $\neg\varphi$. Here $[b, \sigma]$ is the Hamming interval [e.g. Wie87] between b, σ, the union of all the multiple shortest paths between the states (each path corresponds to a different permutation of the variables on which the states disagree).

[5] When b is a full cube, another way to say this is that ψ is b-monotone if it is monotone in the ordinary sense under the translation [Wie87] specified by b.

We shall see in Corollary 2.4.11 that the converse also holds, and so this is indeed a syntactic characterization of when a formula is b-monotone. When this does not hold, the monotonization operator $\mathcal{M}_b(\varphi)$ provides the "closest thing", in the sense that it is the smallest b-monotone ψ s.t. $\varphi \Longrightarrow \psi$. The following result shows that the monotonization can be also be computed based on the syntactic perspective, and $\mathcal{M}_b(\varphi)$ can be efficiently obtained from φ by deleting literals.

Disjunctive form. The monotone overapproximation can be related to a DNF representation of the original formula, a fact that we use extensively in §5.4 and also when we analyze Λ-PDR on specific examples. Starting with a DNF representation of φ, we can derive a DNF representation of $\mathcal{M}_b(\varphi)$ by dropping in each term the literals that agree with b. Intuitively, a "constraint" that $\sigma \models \ell$ in order to have $\sigma \models \mathcal{M}_b(t)$ where ℓ agrees with b is dropped because if $\sigma \models \mathcal{M}_b(t)$ then flipping a bit σ to disagree with b results in a state $\tilde{\sigma}$ such that also $\tilde{\sigma} \models \mathcal{M}_b(t)$, as $\sigma \leq_b \tilde{\sigma}$.

Lemma 2.4.7 (Generalization of Bshouty [Bsh95], Lemma 1(7))**.** *Let $\varphi = t_1 \vee \ldots \vee t_m$ in DNF. Then the monotonization $\mathcal{M}_b(\varphi) \equiv \mathcal{M}_b(t_1) \vee \ldots \vee \mathcal{M}_b(t_m)$ where $\mathcal{M}_b(t_i) \equiv t_i \setminus b = \bigwedge\{\ell \in t_i \wedge \ell \notin b\}$.*

Proof. First we argue that for any term t, $\mathcal{M}_b(t) \equiv t \setminus b$. Denote the rhs by ψ. Let $x \in \mathcal{M}_b(t)$. Then there is $v \models t$ such that $v \leq_b x$. Let $\ell \in t$; $v \models \ell$. Consider a literal $\ell \in \psi$, then $\ell \in t$ and $\ell \notin b$. Since $v \models t$, the former means that in particular $v \models \ell$. The latter means that to satisfy $v \leq_b x$ necessarily v, x agree on the variable in ℓ, and hence also $x \models \ell$. This proves $\mathcal{M}_a(t) \Longrightarrow \psi$. For the other direction, let $x \models \psi$. Let v be obtained from x by setting every variable $p_i \in dom(b)$ that does not appear in ψ to disagree with the corresponding value in b; then $v \leq_a x$. Now $v \models \psi$ (since these variables do not appear in ψ), and, furthermore, $v \models \ell$ for every $\ell \in t$ that was dropped from t to ψ, because v disagrees with b on those literals, which are those that appear in b in a negated form compared to ℓ. Overall, $v \models t$, which implies $x \models \mathcal{M}_a(t)$.

We now claim, more generally, that $\mathcal{M}_b(\psi_1 \vee \psi_2) \equiv \mathcal{M}_b(\psi_1) \vee \mathcal{M}_a(\psi_2)$: Let $x \models \mathcal{M}_b(\psi_1 \vee \psi_2)$. Then there is $v \models \psi_1 \vee \psi_2$ such that $x \leq_b x$. If $v \models \psi_1$, by definition we must have $x \models \mathcal{M}_b(\psi_1)$ and in particular $x \models \mathcal{M}_b(\psi_1) \vee \mathcal{M}_b(\psi_2)$; similarly for ψ_2. This shows $\mathcal{M}_b(\psi_1 \vee \psi_2) \Longrightarrow \mathcal{M}_b(\psi_1) \vee \mathcal{M}_b(\psi_2)$. As for the other direction, let $x \models \mathcal{M}_b(\psi_1) \vee \mathcal{M}_b(\psi_2)$. Without loss of generality, assume $x \models \mathcal{M}_b(\psi_1)$. Then there is $v \models \psi_1$, and in particular $v \models \psi_1 \vee \psi_2$, such that $v \leq_b x$. So we must have $x \models \mathcal{M}_b(\psi_1 \vee \psi_2)$. □

A corollary provides a canonical (inefficient) disjunctive form for $\mathcal{M}_b(\varphi)$:

Corollary 2.4.8. *Given a state v, we denote $cube_b(v) \stackrel{\text{def}}{=} \mathcal{M}_b(v) = \bigwedge\{p \mid v[p] = true, p \notin b\} \wedge \bigwedge\{\neg p_i \mid v[p] = false, \neg p \notin b\}$. Then $\mathcal{M}_b(\varphi) \equiv \bigvee_{v \models \varphi} cube_b(v)$.*

Proof. Apply Lemma 2.4.7 to the representation of φ as the disjunction of all satisfying states. □

In particular, if $v \models \varphi$ then $cube_b(v) \Longrightarrow \mathcal{M}_b(\varphi)$ (follows from Lemma 2.4.7 thinking about the representation $v \vee \varphi$). A similar property holds under the weaker premise that v is known to belong to the monotonization:

Lemma 2.4.9. *If* $v \models \mathcal{M}_b(\varphi)$ *then* $cube_b(v) \Longrightarrow \mathcal{M}_b(\varphi)$.

Proof. From the premise it follows that $cube_b(v) \Longrightarrow \mathcal{M}_b(\mathcal{M}_b(\varphi))$, but by Lemma 2.4.7 and conversions to DNF, $\mathcal{M}_b(\mathcal{M}_b(\varphi)) \equiv \mathcal{M}_b(\varphi)$. □

Another corollary of Lemma 2.4.7 is that the DNF size cannot increase from φ to $\mathcal{M}_b(\varphi)$:

Lemma 2.4.10. $|\mathcal{M}_b(\varphi)|_{\text{dnf}} \leq |\varphi|_{\text{dnf}}$.

Proof. Applying Lemma 2.4.7 to the minimal DNF of φ gives a DNF representation of $\mathcal{M}_b(\varphi)$ with the same number of terms. □

As an aside, at this point we can fulfill our promise for a syntactic characterization of b-monotone formulas:

Corollary 2.4.11. *The converse of Lemma 2.4.6 also holds: If* ψ *is* b*-monotone, then it is expressible in DNF such that every variable* $p \in dom(b)$ *appears in* ψ *in polarity according to* $\neg b[p]$*:* p *appears only positively if* $b[p] = false$ *and only negatively if* $b[p] = true$*. (Variables* $p \notin dom(b)$ *are unconstrained, and can appear both positively and negatively in different terms of the formula.)*

Proof. When ψ is b-monotone, $\mathcal{M}_b(\psi) \equiv \psi$; the DNF representation of Corollary 2.4.8 is as desired. □

2.4.2 Monotone Hull

We now define the monotone hull, which is a conjunction of b-monotone overapproximations over all b from a fixed set of states B. We start with the definition that uses a conjunction over monotone-overapproximations w.r.t. individual states, and then extend this to the union of (partial) cubes.[6]

Definition 2.4.12 (Monotone Hull)**.** *The* monotone hull *of a formula* φ *w.r.t. a set of states* B *is* $\text{MHull}_B(\varphi) = \bigwedge_{b \in B} \mathcal{M}_b(\varphi)$.

The monotone hull can be simplified to use a succinct DNF representation of the basis B instead of a conjunction over all states. (This is the motivation for generalizing $\mathcal{M}_b(\cdot)$ to a cube b in §2.4.)

Lemma 2.4.13. *If* $B = b_1 \vee \ldots \vee b_m$ *and* $b_1, \ldots, b_m$ *are cubes, then* $\text{MHull}_B(\varphi) \equiv \mathcal{M}_{b_1}(\varphi) \wedge \ldots \wedge \mathcal{M}_{b_m}(\varphi)$.

Proof. It follows from the definition that $\text{MHull}_B(\cdot)$ distributes over union in B. Hence, $\text{MHull}_B(\varphi) \equiv \text{MHull}_{b_1}(\varphi) \wedge \ldots \wedge \text{MHull}_{b_m}(\varphi)$. Further, $\text{MHull}_{b_i}(\varphi) = \bigwedge_{\sigma_b \in b_i} \text{MHull}_{\{\sigma_b\}}(\varphi) = \bigwedge_{\sigma_b \in b_i} \mathcal{M}_{\sigma_b}(\varphi)$. It remains to argue that $\mathcal{M}_{b_i}(\varphi) \equiv \bigwedge_{\sigma_b \in b_i} \mathcal{M}_{\sigma_b}(\varphi)$.

[6] The original work [Bsh95] considered only a conjuction w.r.t. individual states; Lemma 2.4.13 is new, and becomes important in Chapter 5, especially for the results of §5.5.

$\subseteq$: By definition, if $\sigma \models \mathcal{M}_{b_i}(\varphi)$ then there is $x \models \varphi$ s.t. $x \leq_{b_i} \sigma$. In particular, $x \leq_{\sigma_b} \sigma$ for every $\sigma_b \models b_i$ (because σ_b agrees with all the literals in b_i), and hence $\sigma \models \mathcal{M}_{\sigma_b}(\varphi)$ for every such σ_b.
$\supseteq$: Suppose that σ is a model of the rhs. Let σ_b be the state obtained from σ by setting the variables present in b_i to be as in b_i (geometrically, this the projection of σ onto the cube b_i). We have $\sigma_b \models b_i$. Thus $\sigma \models \mathcal{M}_{\sigma_b}(\varphi)$, so there exists $x \models \varphi$ such that $x \leq_{\sigma_b} \sigma$. But because σ_b agrees with σ on all literals except those also present in b_i, this implies also that $x \leq_{b_i} \sigma$. Hence also $\sigma \models \mathcal{M}_{b_i}(\varphi)$. □

Note that when $B = b$ is a single cube, $\mathrm{MHull}_b(\varphi) = \mathcal{M}_b(\varphi)$.

Additional lemmas. Before proceeding, we state a few helpful, simple lemmas that we use later:

Lemma 2.4.14. $\varphi \Longrightarrow \mathrm{MHull}_B(\varphi)$.

Proof. $\varphi \Longrightarrow \mathcal{M}_b(\varphi)$ for every $b \in B$, from the definition of b-monotone overapproximation. Hence also $\varphi \Longrightarrow \bigwedge_{b\in B} \mathcal{M}_b(\varphi)$. □

Lemma 2.4.15. *If* $\varphi_1 \Longrightarrow \varphi_2$ *then* $\mathrm{MHull}_b(\varphi_1) \Longrightarrow \mathrm{MHull}_b(\varphi_2)$.

Proof. $\mathcal{M}_b(\varphi_1) \Longrightarrow \mathcal{M}_b(\varphi_2)$ for every $b \in B$ by Lemma 2.4.5, so if $\sigma \models \bigwedge_{b\in B} \mathcal{M}_b(\varphi_1)$ it also satisfies $\sigma \models \bigwedge_{b\in B} \mathcal{M}_b(\varphi_2)$. □

Lemma 2.4.16. *The monotone hull is idempotent, that is,* $\mathrm{MHull}_B(\mathrm{MHull}_B(\varphi)) \equiv \mathrm{MHull}_B(\varphi)$.

Proof. We claim that for every $b \in B$, $\mathcal{M}_b(\mathrm{MHull}_B(\varphi)) \equiv \mathcal{M}_b(\varphi)$, which implies $\mathrm{MHull}_B(\mathrm{MHull}_B(\varphi)) = \bigwedge_{b\in B} \mathcal{M}_b(\mathrm{MHull}_B(\varphi)) \equiv \bigwedge_{b\in B} \mathcal{M}_b(\varphi) = \mathrm{MHull}_B(\varphi)$.

Let $b \in B$. $\mathcal{M}_b(\varphi) \subseteq \mathcal{M}_b(\mathrm{MHull}_B(\varphi))$ because $\varphi \subseteq \mathrm{MHull}_B(\varphi)$ (Lemma 2.4.14) and by Lemma 2.4.15. Since from the definition $\mathrm{MHull}_B(\varphi) \subseteq \mathcal{M}_b(\varphi)$, again by Lemma 2.4.16 $\mathcal{M}_{\subseteq}(\mathrm{MHull}_B(\varphi))\mathcal{M}_b(\mathcal{M}_b(\varphi))$ and $\mathcal{M}_b(\cdot)$ is idempotent from the definition as least b-monotone overapproximation. □

2.4.3 Monotone Basis and Monotone Span

In general, $\mathrm{MHull}_B(\varphi)$ is not equivalent to φ. However, we can always choose B so that $\mathrm{MHull}_B(\varphi) \equiv \varphi$. A set B that suffices for this is called a basis:

Definition 2.4.17 (Monotone Basis [Bsh95])**.** *A* monotone basis *is a set of states B. It is a basis* for a formula φ *if* $\varphi \equiv \mathrm{MHull}_B(\varphi)$.

Conversely, given a set B, we are interested in the set of formulas for which B forms a basis:

Definition 2.4.18 (Monotone Span)**.** $\mathrm{MSpan}(B) = \{\mathrm{MHull}_B(\varphi) \mid \varphi \text{ over } \Sigma\}$*, the set of formulas for which B is a monotone basis.*

The following theorem provides a syntactic characterization of $\mathrm{MSpan}(B)$, as the set of all formulas that can be written in CNF using clauses that exclude states from the basis:

Theorem 2.4.19 (Bshouty [Bsh95], Lemma 4)**.** *$\varphi \in \mathrm{MSpan}(B)$ iff there exist clauses $c_1, \ldots, c_s$ such that $\varphi \equiv c_1 \wedge \ldots \wedge c_s$ and for every $1 \leq i \leq s$ there exists $b_j \in B$ such that $b_j \not\models c_i$.*

(We defer the proof to §5.2, where this result is obtained from a conjunctive characterization of the monotone hull that we obtain in Thm. 5.2.1.)

In particular, every φ admits some monotone basis B, that can be constructed by writing a CNF representation of φ and choosing for B a state $b_j \not\models c_j$ for each clause c_j.

A useful consequence of this characterization is that the monotone basis is closed under conjunction:

Lemma 2.4.20. *If $\varphi_1, \varphi_2 \in \mathrm{MSpan}(B)$ then also $\varphi_1 \wedge \varphi_2 \in \mathrm{MSpan}(B)$.*

Proof. By Thm. 2.4.19 and the premise, $\varphi_1 \equiv c_1 \wedge \ldots \wedge c_s$ and $\varphi \equiv c'_1 \wedge \ldots \wedge c'_{s'}$ where each c_i and c'_i excludes some state from B. Hence $\varphi_1 \wedge \varphi_2 \equiv c_1 \wedge \ldots \wedge c_s \wedge c'_1 \wedge \ldots \wedge c'_{s'}$, which by the other direction of Thm. 2.4.19 means that $\varphi_1 \wedge \varphi_2 \in \mathrm{MSpan}(B)$, as desired. □

2.4.4 Exact Learning Using the Monotone Theory

The monotone theory was first developed by Bshouty for the purpose of exact learning formulas that are not monotone. Essentially, the idea is to reconstruct the formula φ by finding a monotone basis $B = \{b_1, \ldots, b_t\}$ for it, and constructing $\mathrm{MHull}_B(\varphi)$ while using equivalence and membership queries. The original paper [Bsh95] includes two algorithms: 1. The Λ-algorithm (Alg. 5), which can learn a formula provided that the monotone basis is known a-priori, such as when the formulas is known to be almost monotone ($O(1)$ of the terms contain negated variables), and 2. The CDNF algorithm (Alg. 6), which gradually finds a monotone basis, and is proven efficient for learning CDNF formulas—formulas that have both a short CNF and a short DNF representation. The code for the algorithms is shown here, but we omit their precise analysis. The Λ-algorithm is the basis of an algorithm for learning almost-monotone invariants in §4.4, and the analysis there follows the analysis of the original learning algorithm (while also factoring reachability considerations). The CDNF algorithm inspires our algorithm for inferring CDNF invariants in §4.6, but in this case the inference algorithm departs from the learning algorithm in significant ways that we highlight there.

Algorithm 5 Λ-algorithm: exact learning with a known monotone basis [Bsh95]

Assuming a known basis $\{b_1, \ldots, b_t\}$ (Def. 2.4.17)
procedure Λ-ALGORITHM
    $H_1, \ldots, H_t \leftarrow \mathit{false}$
    while $\sigma' \leftarrow$ EquivalenceQuery $\left(\bigwedge_{i=1}^{t} H_i\right)$ is not $\perp$ **do**
        for $i = 1, \ldots, t$ **do**
            if $\sigma' \not\models H_i$ **then**
                $d \leftarrow$ MONGENMEM(σ', b_i)
                $H_i \leftarrow H_i \vee d$
    return $\bigwedge_{i=1}^{t} H_i$

procedure MONGENMEM(σ, b)
    if MembershipQuery $(x) \neq \mathit{true}$ **then**
        fail (not a basis)
    $v \leftarrow \sigma$
    walked $\leftarrow \mathit{true}$
    while walked **do**
        walked $\leftarrow \mathit{false}$
        for $j = 1, \ldots, n$ such that $b[p_j] \neq v[p_j]$ **do**
            $x \leftarrow v[p_j \mapsto b[p_j]]$
            if MembershipQuery $(x) = \mathit{true}$ **then**
                $v \leftarrow$ x
                walked $\leftarrow \mathit{true}$
    return $\mathit{cube}_b(v)$

Algorithm 6 CDNF-algorithm: exact learning CDNF formulas [Bsh95]

procedure CDNF-ALGORITHM
    $t \leftarrow 0$
    while $x \leftarrow$ EquivalenceQuery $\left(\bigwedge_{i=1}^{t} H_i\right)$ is not $\perp$ **do**
        if $x \models \bigwedge_{i=1}^{t} H_i$ **then**
            $H_{t+1} \leftarrow \mathit{false}$; $b_{t+1} \leftarrow x$; $t \leftarrow t + 1$
        else
            $\sigma' \leftarrow x$
            for $i = 1, \ldots, t$ **do**
                if $\sigma' \not\models H_i$ **then**
                    $d \leftarrow$ MONGENMEM(σ', b_i)
                    $H_i \leftarrow H_i \vee d$
    return $\bigwedge_{i=1}^{t} H_i$

Figure 2.1: Exact learning algorithms using the monotone theory.

Chapter 3

Learning Models and Lower Bounds for Invariant Inference

This chapter is based on the results published in [FISS20].

In this chapter we analyze the complexity of polynomial-length invariant inference in a learning model, called *the Hoare-query model*, which is general enough to capture algorithms such as IC3/PDR and its variants. An algorithm in this model learns about the system's reachable states by querying the validity of Hoare triples.

We show that in general an algorithm in the Hoare-query model requires an exponential number of queries. Our lower bound is information-theoretic and applies even to computationally unrestricted algorithms, showing that no choice of generalization from the partial information obtained in a polynomial number of Hoare queries can lead to an efficient invariant inference procedure in this class.

We then show, for the first time, that by utilizing rich Hoare queries, as done in PDR, inference can be exponentially more efficient than approaches such as ICE learning, which only utilize inductiveness checks of candidates. We do so by constructing a class of transition systems for which a simple version of PDR with a single frame infers invariants in a polynomial number of queries, whereas every algorithm using only inductiveness checks and counterexamples requires an exponential number of queries.

Our results also shed light on connections and differences with the classical theory of exact concept learning with queries, and imply that learning from counterexamples to induction is harder than classical exact learning from labeled examples. This demonstrates that the convergence rate of Counterexample-Guided Inductive Synthesis depends on the form of counterexamples.

```
1  init  x_1 = ... = x_n = 0
2  axiom  ∃!i, 1 ≤ i ≤ n.  c_i = 1
3
4  function add-double(a,b) = (a + 2 · b) mod 2^n
5
6  while *
7      input  y_1, ..., y_n
8      if  c_1 :
9          (x_1, x_2, ..., x_{n-1}, x_n)      := add-double((x_1, x_2, ..., x_{n-1}, x_n),(y_1, y_2, ..., y_{n-1}, y_n))
10     if  c_2 :
11         (x_2, x_3, ..., x_n, x_1)          := add-double((x_2, x_3, ..., x_n, x_1),(y_2, y_3, ..., y_n, y_1))
12     ...
13     if  c_n :
14         (x_n, x_1, ..., x_{n-2}, x_{n-1})  := add-double((x_n, x_1, ..., x_{n-2}, x_{n-1}),(y_n, y_1, ..., y_{n-2}, y_{n-1}))
15     assert  ¬(x_1 = ... = x_n = 1)
```

Figure 3.1: An example propositional transition system for which we would like to infer an inductive invariant. The state is over $x_1, \ldots, x_n$. The variables $y_1, \ldots, y_n$ are inputs and can change arbitrarily in each step. $c_1, \ldots, c_n$ are immutable, with the assumption that exactly one is true.

3.1 Overview

Coming up with inductive invariants is one of the most challenging tasks of formal verification—it is often referred to as the *"Eureka!"* step. This thesis studies the asymptotic complexity of automatically inferring CNF invariants of polynomial length, a problem we call ***polynomial-length inductive invariant inference***, in a SAT-based black-box model.

Consider the dilemmas Abby faces when she attempts to develop an algorithm for this problem from first principles. Abby is excited about the popularity of SAT-based inference algorithms. Many such algorithms operate by repeatedly performing checks of Hoare triples of the form $\{\alpha\}\delta\{\beta\}$, where α, β are a precondition and postcondition (resp.) chosen by the algorithm in each query, and δ is the given transition relation (loop body). A SAT solver implements the check. We call such checks ***Hoare queries***, and focus in this chapter on *black-box* inference algorithms in the ***Hoare-query model***: algorithms that access the transition relation solely through Hoare queries.

Fig. 3.1 displays one example program that Abby is interested in inferring an inductive invariant for. In this program, a number $\mathbf{x}$, represented by n bits, is initialized to zero, and at each iteration incremented by an even number that is decided by the input variables $\mathbf{y}$ (all computations are mod 2^n). The representation of the number $\mathbf{x}$ using the bits $x_1, \ldots, x_n$ is determined by another set of bits $c_1, \ldots, c_n$, which are all immutable, and only one of them is true: if $c_1 = \mathit{true}$, the number is represented by $x_1, x_2, \ldots, x_n$, if $c_2 = \mathit{true}$ the least-significant bit (lsb) shifts and the representation is $x_2, x_3, \ldots, x_n, x_1$ and so on. The safety property is that $\mathbf{x}$ is never equal to the number with all bits 1. Intuitively, this holds because the number $\mathbf{x}$ is always even. An *inductive invariant* states this fact, taking into account the differing representations, by stating that the lsb (as chosen by $\mathbf{c}$) is always 0: $I = (c_1 \to \neg x_1) \wedge \ldots (c_n \to \neg x_n)$. Of course, Abby aims to verify many systems, of which Fig. 3.1 is but one example.

Algorithm 7
Backward reachability

```
1: procedure BACKWARD-REACH(δ)
2:     I ← ¬Bad
3:     while {I}δ{I} not valid do
4:         σ, σ' ← CTI(δ, I)
5:         d ← BLOCK(δ,σ)
6:         I ← I ∧ ¬d
       return I
```

Algorithm 8 Exact block

```
1: procedure BLOCK-CUBE(δ,σ)
       return ⋀{p_i | σ ⊨ p_i}∧
2:                ⋀{¬p_i | σ ⊨ ¬p_i}
```

Algorithm 9
Block with generalization from init-step reachability

```
1: procedure BLOCK-PDR-1(δ, σ)
2:     d ← cube(σ)
3:     for l ∈ cube(σ) do
4:         t ← d \ {l}
5:         if (Init ⟹ ¬t) ∧ {Init}δ{¬t} then
6:             d ← t
       return d
```

3.1.1 Example: Backward Reachability with Generalization

How should Abby's algorithm go about finding inductive invariants? One known strategy is that of *backward reachability*, in which the invariant is strengthened to exclude states from which bad states may be reachable.[1] Alg. 7 is an algorithmic backward reachability scheme: it repeatedly checks for the existence of a counterexample to induction (a transition σ, σ' of δ from $\sigma \models I$ to $\sigma' \not\models I$), and strengthens the invariant to exclude the pre-state σ using the formula BLOCK returns.

Alg. 7 depends on the choice of BLOCK. The most basic approach is of Alg. 8, which excludes *exactly* the pre-state, by conjoining to the invariant the negation of the cube of σ (the conjunction of all literals that hold in the state; the only state that satisfies $cube(\sigma)$ is σ itself). For example, when Alg. 7 needs to block the state $\mathbf{x} = 011\ldots1, \mathbf{c} = 000\ldots1$ (this state reaches the bad state $\mathbf{x} = 111\ldots1, \mathbf{c} = 000\ldots1$), Alg. 8 does so by conjoining to the invariant the negation of $\neg x_n \wedge x_{n-1} \wedge x_{n-2} \wedge \ldots x_1 \wedge \neg c_n \wedge \neg c_{n-1} \wedge \neg c_{n-2} \wedge \ldots c_1$.

Alas, Alg. 7 with blocking by Alg. 8 is not efficient. In essence it operates by enumerating and excluding the states that are backward-reachable from bad. The number of such states is potentially exponential, making Alg. 8 unsatisfactory. For instance, the example of Fig. 3.1 requires the exclusion of all states in which $\mathbf{x}$ is odd for every choice of lsb, a number of states exponential in n. The algorithm would thus require an exponential number of queries to arrive at a (CNF) inductive invariant, even though a CNF invariant with only n clauses exists (as above).

Efficient inference hence requires Abby to exclude more than a single state at each time, namely, to *generalize* from a counterexample—as real algorithms do. What generalization strategy could Abby choose that would lead to efficient invariant inference?

[1] Our results are not specific to backward reachability algorithms; we use them here for motivation and illustration.

3.1.2 All Generalizations Are Wrong

One simple generalization strategy Abby considers appears in Alg. 9, based on the standard ideas in IC3/PDR [Bra11, EMB11] and subsequent developments [e.g. HB12, KGC14]. It starts with the cube of the state (as Alg. 8) and attempts to drop some of the literals, resulting in a smaller conjunction satisfied by more states; all these states are excluded from the candidate in line 6 of Alg. 7. Hence with this generalization Alg. 7 can exclude many states in each iteration, overcoming the problem with the naive algorithm above. Alg. 9 chooses to drop a literal from the conjunction if no state reachable in at most one step from *Init* satisfies the conjunction even when that literal is omitted (line 4 of Alg. 9); we refer to this algorithm as PDR-1, since it resembles PDR with a single frame.

For example, when in the example of Fig. 3.1 the algorithm attempts to block the state with $\mathbf{x} = 011\ldots1, \mathbf{c} = 000\ldots1$, Alg. 9 minimizes the cube to $d = x_1 \wedge c_1$, because no state reachable in at most one step satisfies d, but this is no longer true when another literal is omitted. Conjoining the invariant with $\neg d$ (in line 6 of Alg. 7) produces a clause of the invariant, $c_1 \rightarrow \neg x_1$. In fact, our results show that PDR-1 finds the aforementioned invariant in n^2 queries.

Yet there is a risk in over-generalization, that is, of dropping *too many* literals and excluding too many states. In Alg. 7, generalization must not return a formula that some reachable states satisfy, or else the candidate I would exclude reachable states, and would not be an inductive invariant. Alg. 9 chooses to take the strongest conjunction that does not exclude any state reachable in at most one step; it is of course possible (and plausible) that some states are reachable in two steps but not in one. Alg. 7 with the generalization in Alg. 9 might fail in such cases.

The necessity of generalization, on the one hand, and the problem of over-generalization on the other leads in practice to complex heuristic techniques. Instead of simple backward reachability with generalization per Alg. 7, PDR never commits to a particular generalization [EMB11] through a sequence of *frames*, which are (in some sense) a sequence of candidate invariants. We study the frames in detail in Chapter 5; Alg. 9 corresponds to generalization in the first frame.

Overall, the study of backward reachability and the PDR-1 generalization leaves us with the question: ***Is there a choice of generalization that can be used—in any way—to achieve an efficient invariant inference algorithm?***

In a non-interesting way, the answer is *yes*, there is a "good" way to generalize. Use Alg. 7, with the following generalization strategy: upon blocking a pre-state σ, compute an inductive invariant of polynomial length, and return the clause of the invariant that excludes σ,[2] and this terminates in a polynomial number of steps, assuming that the invariant has a polynomially-long CNF representation.

Such generalization is clearly unattainable. It requires (1) perfect information of the transition system, and (2) solving a computationally hard problem, since we show that polynomial-length

[2]Such a clause exists because σ is backward reachable from bad states, and thus excluded from the invariant.

inference is Σ_2^P-hard (Thm. 3.2.3). What happens when generalization is computationally unbounded (an arbitrary function), but operates based on *partial information* of the transition system? Is there a ***generalization from partial information***, be it computationally intractable, that facilitates ***efficient inference***? If such a generalization exists we may wish to view invariant inference heuristics as *approximating* it in a computationally efficient way.

Similar questions arise in interpolation-based algorithms, only that generalization is performed not from a concrete state, but from a bounded unreachability proof. Still it is challenging to generalize enough to make progress but not too much as to exclude reachable states (or include states from which bad is reachable in algorithms that gradually add states to the candidate).

Our Results

Our first main result in this chapter is that ***in general, there does not exist a generalization scheme from partial information*** leading to efficient inference based on Hoare queries. Technically, we prove that even a computationally unrestricted generalization from information gathered from Hoare queries requires an exponential number of queries. This result applies to any generalization strategy (and any algorithm using it) that can be modeled using Hoare queries, including Alg. 7 as well as more complex algorithms such as PDR. We also extend this lower bound to a model capturing interpolation-based algorithms (Thm. 3.4.7).

These results are surprising because a-priori it would seem possible, using unrestricted computational power, to devise queries that repeatedly halve the search space, yielding an invariant with a polynomial number of queries (the number of candidates is only exponential because we are interested in invariants up to polynomial length). We show that this is impossible to achieve using Hoare queries.

3.1.3 Inference Using Rich Queries

So far we have established strong impossibility results for invariant inference based on Hoare queries in the general case, even with computationally unrestricted generalization. We now turn to shed some light on the techniques that inference algorithms such as PDR employ in practice. One of the fundamental principles of PDR is the *incremental construction* of invariants relying on *rich Hoare queries*. PDR-1 demonstrates a simplified realization of this principle. When PDR-1 considers a clause to strengthen the invariant, it checks the reachability of that individual clause from *Init*, rather than the invariant as a whole. This is the Hoare query $\{Init\}\delta\{\neg t\}$ in line 4 of Alg. 9, in which, crucially, the precondition is different from the postcondition. The full-fledged PDR is similar in this regard, strengthening a frame according to reachability from the previous frame via relative induction checks [Bra11] (see also Chapter 5).

The algorithm in Alg. 7 + Alg. 8 is fundamentally different, and uses only inductiveness queries $\{I\}\delta\{I\}$, a specific form of Hoare queries where the precondition and postcondition are the same. Algorithms performing only inductiveness checks can in fact be very sophisticated, traversing the domain of candidates in clever ways. This approach was formulated in the *ICE*

learning framework for learning inductive invariants [GLMN14, GNMR16], in which algorithms present new candidates based on positive, negative, and implication examples returned by a "teacher" in response to incorrect candidate invariants.[3] The main point is that such algorithms do not perform queries other than inductiveness, and choose the next candidate invariant based solely on the counterexamples to induction showing that the previous candidates were unsuitable.

The contrast between the two approaches raises the question: ***Is there a benefit to invariant inference in Hoare queries richer than inductiveness?*** For instance, to model PDR in the ICE framework, Vizel et al. [VGSM17] extended the framework with relative inductiveness checks, but the whether such an extension is necessary remained an open question.

Our Results

Our second significant result in this chapter is showing an exponential gap between the general Hoare-query model and the more specific inductiveness-query model. To this end, we construct a class of transition systems, including the example of Fig. 3.1, for which (1) PDR-1, which is a Hoare-query algorithm, infers an invariant in a polynomial number of queries, but (2) *every* inductiveness-query algorithm requires an *exponential* number of queries, that is, an exponential number of candidates before it finds a correct inductive invariant. This demonstrates that analyzing the reachability of clauses separately can offer an exponential advantage in certain cases. This also proves that PDR cannot be cast in the ICE framework, and that the extension by Vizel et al. [VGSM17] is necessary and strictly increases the power of inference with a polynomial number of queries. To the best of our knowledge, this is not only the first lower bound on ICE learning demonstrating such an exponential gap (also see the discussion in §3.7), but also the first polynomial upper bound on PDR for a class of systems (see also the discussion in §5.9).

We show this separation on a class of systems constructed using a technical notion of *maximal systems for antimonotone invariants*. These are systems for which there exists an antimonotone invariant (namely, a CNF invariant where propositional variables appear only negatively) with a linear number of clauses, and the transition relation includes *all* the transitions that are allowed by this invariant. For example, a maximal system can easily be constructed from Fig. 3.1: this system allows every transition between states satisfying the invariant (namely, between all even $\mathbf{x}$'s with the same representation), and also every transition between states violating the invariant (namely, between all odd $\mathbf{x}$'s with the same representation)[4]; a maximal system also includes all the transitions from states that violate that invariant to the states that satisfy it (here, between odd $\mathbf{x}$ and even $\mathbf{x}$ with the same $\mathbf{c}$). The success of PDR-1 on such systems relies on the small diameter (every reachable state is reachable in one step) and harnesses properties of prime consequences of antimonotone formulas. In contrast, we show that for inductiveness-query

[3] Our formulation focuses on implication examples—counterexamples to inductiveness queries—and strengthens the algorithm with full information about the set of initial and bad states instead of positive and negative examples (resp.).

[4] Transitions violating the $\mathbf{c}$ axiom or modifying it are excluded in this modeling.

algorithms this class is as hard as the class of *all* programs admitting antimonotone invariants, whose hardness is established from the results of §3.1.2. For example, from the perspective of inductiveness-query algorithms, the example of Fig. 3.1, which is a maximal program as explained above, is as hard as *any* system that admits its invariant (and also respects the **c** axiom and leaves **c** unchanged). This is because an inductiveness-query algorithm can only benefit from having *fewer* transitions and hence fewer counterexamples to induction, whereas maximal programs include as many transitions as possible. If an inductiveness query algorithm is to infer an invariant for the example of Fig. 3.1, it must also be able to infer an invariant for all systems whose transitions are a *subset* of the transitions of this example. This includes systems with an exponential diameter, as well as systems admitting other invariants, potentially exponentially long. This program illustrates our lower bound construction, which takes *all* maximal programs for monotone-CNF invariants.

In our lower bound we follow the existing literature on the analysis of inductiveness-query algorithms, which focuses on the worst-case notion w.r.t. potential examples (strong convergence in Garg et al. [GLMN14]). An interesting direction is to analyze inductiveness-query algorithms that exercise some control over the choice of counterexamples to induction, or under probabilistic assumptions on the distribution of counterexamples.

3.1.4 A Different Perspective: Exact Learning of Invariants with Hoare Queries

This chapter develops a theory of exact learning of inductive invariants with Hoare queries, akin to the classical theory of concept learning with queries [Ang87b]. The results outlined above are consequences of natural questions about this model: The impossibility of generalization from partial information (§3.1.2) stems from an exponential lower bound on the Hoare-query model. The power of rich Hoare queries (§3.1.3) is demonstrated by an exponential separation between the Hoare- and inductiveness-query models, in the spirit of the gap between concept learning using both equivalence and membership queries and concept learning using equivalence queries alone [Ang90].

The similarity between invariant inference (and synthesis in general) and exact concept learning has been observed before [e.g. JGST10, GLMN14, JS17, ABD+15, BDVY17]. Our work highlights some interesting differences and connections between invariant learning with Hoare, and concept learning with equivalence and membership queries. This comparison yields our third significant result in this chapter: the impossibility of translating algorithms from concept learning with queries to invariant inference with queries. Our fourth significant result is a proof that learning from counterexamples to induction is inherently harder than learning from examples labeled as positive or negative, formally corroborating the intuition advocated by Garg et al. [GLMN14]. More broadly, the complexity difference between learning from labeled examples and learning from counterexamples to induction demonstrates that the convergence rate of learning in Counterexample-Guided Inductive Synthesis [e.g. STB+06, JGST10, JS17]

depends on the form of examples. The proof of this result builds on the lower bounds discussed earlier, and is discussed in §3.6.[5]

3.2 Polynomial-Length Invariant Inference

In this section we formally define the problem of *polynomial-length invariant inference* for CNF formulas, which is the focus of this chapter, and discuss its complexity. We then relate the problem to the problem of inferring DNF formulas with polynomially many term via the forward-backward duality (§2.2.1), and focus on the case of CNF in the rest of the chapter.

Definition 3.2.1 (Inductive Invariant Inference from Class of Invariants)**.** *For a class of transition systems $\mathcal{P}$ and a class of invariants $\mathcal{L}$,* inductive invariant inference *is the problem: Given a transition system $TS \in \mathcal{P}$ over Σ,* decide *whether there exists an inductive invariant $I \in \mathcal{L}$ for TS.*

Our object of study is the problem of polynomial-length inference, which is invariant inference with $\mathcal{L}$ being the set of CNF formulas of polynomial size:

Definition 3.2.2 (Polynomial-Length Inductive Invariant Inference)**.** *The* polynomial-length inductive invariant inference problem *for a class of transition systems $\mathcal{P}$ and a polynomial $p(n) = \Omega(n)$ is the problem: Given a transition system $TS \in \mathcal{P}$ over Σ,* decide *whether there exists an inductive invariant $I \in \mathrm{CNF}_{p(n)}$ for TS, where $n = |\Sigma|$.*

Unless otherwise stated, throughout this chapter, by "invariant inference" we refer to polynomial-length inductive invariant inference.

Notation. In the sequel, when considering the polynomial-length inductive invariant inference problem of a transition system $TS = (Init, \delta, Bad) \in \mathcal{P}$, we denote by Σ the vocabulary of $Init$, Bad and δ. Further, we denote $n = |\Sigma|$.

Complexity

The *complexity* of polynomial-length inference is measured in $|TS| = |Init| + |\delta| + |Bad|$. Note that the invariants are required to be polynomial in $n = |\Sigma|$.

$\mathrm{CNF}_{p(n)}$ is a rich class of invariants. Inference in more restricted classes can be solved efficiently. For example, when only conjunctive candidate invariants are considered, and $\mathcal{P}$ is the set of all propositional transition systems, the problem can be decided in a polynomial number of SAT queries through the Houdini algorithm [FL01, LQ09]. Similar results hold also for CNF formulas with a constant number of literals per clause (by defining a new predicate for each of

[5] It may also be interesting to note that one potential difference between classical learning and invariant inference, mentioned by Löding et al. [LMN16], does not seem to manifest in the results discussed in §3.1.2: the transition systems in the lower bound for inductiveness queries in Corollary 3.5.11 have a *unique* inductive invariant, and still the problem is hard.

the polynomially-many possible clauses and applying Houdini), and for CNF formulas with a constant number of clauses (by translating them to DNF formulas with a constant number of literals per term and applying the dual procedure). However, a restricted class of invariants may miss invariants for some programs and reduces the generality of the verification procedure. Hence in this thesis we are interested in the richer class of polynomially-long CNF invariants. In this case the problem is no longer tractable even with a SAT solver:

Theorem 3.2.3. *Let $\mathcal{P}$ be the set of all propositional transition systems. Then polynomial-length inference for $\mathcal{P}$ is Σ_2^P-complete, where $\Sigma_2^P = NP^{SAT}$ is the second level of the polynomial-time hierarchy.*

We defer the proof to §3.4.1.

We note that polynomial-length inference can be encoded as specific instances of template-based inference; the Σ_2^P-hardness proof of Lahiri and Qadeer [LQ09] uses more general templates and therefore does not directly imply the Σ_2^P-hardness of polynomial-length inference. Lower bounds on polynomial-length inference entail lower bounds on template-based inference.

Remark 3.2.4. *The complexity of inferring DNF invariants can be worked out from the complexity of CNF invariants.*

Lemma 3.2.5. *The complexity of the inference of $\mathcal{P}$ w.r.t. $\mathcal{L} = \mathrm{CNF}_{p(n)}$ is the same as $\mathcal{P}^*$ w.r.t. $\mathcal{L} = \mathrm{DNF}_{p(n)}$.*

Proof. Apply the duality of §2.2.1 to the best algorithm in each setting. □

Remark 3.2.6. *In the above formulation, an efficient procedure for deciding* safety *does not imply polynomial-length inference is tractable, since the program may be safe, but all inductive invariants may be too long. To overcome this technical quirk, we can consider a* promise problem *[Gol06] variant of polynomial-length inference:*

Given a transition system $TS \in \mathcal{P}$,

- ***(Completeness)** If TS has an inductive invariant $I \in \mathrm{CNF}_{p(n)}$, the algorithm must return* yes.
- ***(Soundness)** If TS is not safe the algorithm must return* no.

Other cases, including the case of safety with an invariant outside $\mathrm{CNF}_{p(n)}$, are not constrained. An algorithm deciding safety thus solves also this problem. All the results of this chapter apply both to the standard version above and the promise problem: upper bounds on the standard version trivially imply upper bounds on the promise problem, and in our lower bounds we use transition systems that are either (i) safe and *have an invariant in $\mathrm{CNF}_{p(n)}$, or (ii) unsafe.*

3.3 Invariant Inference with Queries and the Hoare Query Model

In this chapter we study algorithms for polynomial-length inference through *black-box* models of *inference with queries*. In this setting, the algorithm accesses the transition relation through (rich) queries, but cannot read the transition relation directly. Our main model is of *Hoare-query algorithms*, which query the validity of a postcondition from a precondition in one step of the system. Hoare-query algorithms faithfully capture a large class of SAT-based invariant inference algorithms, including PDR and related methods.

A black-box model of inference algorithms facilitates an analysis of the *information* of the transition relation the algorithm acquires. The advantage is that such an information-based analysis sidesteps open computational complexity questions, and therefore results in unconditional lower bounds on the computational complexity of SAT-based algorithms captured by the model. An information-based analysis is also necessary for questions involving unbounded computational power and restricted information, in the context of computationally-unrestricted bounded-reachability generalization (see §3.4.3).

In this section we define the basic notions of queries and query-based inference algorithms. We also define the primary query model we study in the chapter: the Hoare-query model. We introduce and study additional query models in subsequent chapters—the interpolation-query model in §3.4.2, and the inductiveness-query model in §3.5.1.

Inference with queries

We model queries of the transition relation in the following way: A *query oracle* Q is an oracle that accepts a transition relation δ, as well as additional inputs, and returns some output. The additional inputs and the output, together also called the *interface* of the oracle, depend on the query oracle under consideration. A *family* of query oracles $\mathcal{Q}$ is a set of query oracles with the same interface. We consider several different query oracles, representing different ways of obtaining information about the transition relation.

Definition 3.3.1 (Inference algorithm in the query model)**.** *An* inference algorithm from queries*, denoted* $\mathcal{A}^{Q}(Init, Bad, [\delta])$*, is defined w.r.t. a query oracle* Q *and is given:*

- *access to the query oracle* Q*,*
- *the set of initial states (Init) and bad states (Bad);*
- *the transition relation* δ*, encapsulated—hence the notation* $[\delta]$*—meaning that the algorithm cannot access* δ *(not even read it) except for extracting its vocabulary;* δ *can only be passed as an argument to the query oracle* Q*.*

$\mathcal{A}^{Q}(Init, Bad, [\delta])$ *solves the problem of polynomial-length invariant inference for* $(Init, \delta, Bad)$*.*

The Hoare-query model

Our main object of study in this chapter is the Hoare-query model of invariant inference algorithms. It captures SAT-based invariant inference algorithms querying the behavior of a single step of the transition relation at a time.

Definition 3.3.2 (Hoare-Query Model)**.** *For a transition relation δ and input formulas $\alpha, \beta \in \mathrm{wff}(\Sigma)$, the* Hoare-query oracle*, $\mathcal{H}(\delta, \alpha, \beta)$, returns false if $(\alpha \wedge \delta \wedge \neg\beta') \in \mathrm{SAT}$; otherwise it returns true.*

An algorithm in the Hoare-query model*, also called a* Hoare-query algorithm*, is an inference from queries algorithm expecting the Hoare query oracle.*

Intuitively, a Hoare-query algorithm gains access to the transition relation, δ, exclusively by repeatedly choosing $\alpha, \beta \in \mathrm{wff}(\Sigma)$, and calling $\mathcal{H}(\delta, \alpha, \beta)$.

If we are using a SAT solver to compute the Hoare-query, $\mathcal{H}(\delta, \alpha, \beta)$, then when the answer is *false*, the SAT solver will also produce a counterexample pair of states σ, σ' such that $\sigma, \sigma' \models \alpha \wedge \delta \wedge \neg\beta'$. We observe that using binary search, a Hoare-query algorithm can do the same:

Lemma 3.3.3. *Whenever $\mathcal{H}(\delta, \alpha, \beta) = \mathit{false}$, a Hoare-query algorithm can find σ, σ' such that $\sigma, \sigma' \models \alpha \wedge \delta \wedge \neg\beta'$ using $n = |\Sigma|$ Hoare queries.*

Proof. For each $x_i \in \Sigma \uplus \Sigma'$, if $x_i \in \Sigma$, conjoin it to α, else to β, and check whether $\mathcal{H}(\delta, \alpha_i, \beta_i)$ is still *false*. If it is, continue to x_{i+1}; otherwise flip x_i and continue to x_{i+1}. □

Example: PDR as a Hoare-query algorithm The Hoare-query model captures the prominent PDR algorithm, facilitating its theoretical analysis. In general, PDR accesses the transition relation via checks of unreachability in one step and counterexamples to those checks. These operations are captured in the Hoare query model by checking $\mathcal{H}(\delta, F, \alpha)$ or $\mathcal{H}(\delta, F \wedge \alpha, \alpha)$ (for the algorithm's choice of $F, \alpha \in \mathrm{wff}(\Sigma)$), and obtaining a counterexample using a polynomial number of Hoare queries, if one exists (Lemma 3.3.3). Furthemore, the Hoare-query model is general enough to express a broad range of PDR variants that differ in the way they use such checks but still access the transition relation only through such queries.[6] Alg. 10 casts the basic variant from Alg. 1 as a Hoare-query algorithm. The call to MODEL invokes the procedure of Lemma 3.3.3.

[6] A notable exception is ternary simulation [EMB11], which is not a SAT-based operation. However, see Remark 3.4.4.

Algorithm 10 PDR as a Hoare-query algorithm

procedure PDR(*Init*, *Bad*, $[\delta]$)
  $\mathcal{F}_0^{\text{pdr}} \leftarrow \mathit{Init}$
  $N \leftarrow 0$
  while $\forall 1 \le i \le N.\ \mathcal{F}_i^{\text{pdr}} \not\Longrightarrow \mathcal{F}_{i-1}^{\text{pdr}}$ **do**
    $\mathcal{F}_{N+1}^{\text{pdr}} \leftarrow \mathit{true}$
    while $\mathcal{F}_{N+1}^{\text{pdr}} \not\Rightarrow \neg \mathit{Bad}$ **do**
      for $\sigma_b \in \mathcal{F}_{N+1}^{\text{pdr}} \wedge \mathit{Bad}$ **do**
        BLOCK(σ_b, $N+1$)
    $N \leftarrow N+1$
  return $\mathcal{F}_i^{\text{pdr}}$ such that $\mathcal{F}_i^{\text{pdr}} \Longrightarrow \mathcal{F}_{i-1}^{\text{pdr}}$

procedure BLOCK(σ_b, i)
  if $i = 0$ **then**
    unsafe
  while $\mathcal{H}(\delta, \mathcal{F}_{i-1}^{\text{pdr}}, \neg\sigma_b) = \mathit{false}$ or $\mathcal{F}_{i-1}^{\text{pdr}} \not\Longrightarrow \neg\sigma_b$ **do**
    $(\sigma, \sigma') \leftarrow \text{MODEL}([\delta], \mathcal{F}_i^{\text{pdr}}, \sigma_b)$ or $\sigma \leftarrow \sigma_b$ if $\sigma_b \not\models \mathcal{F}_{i-1}^{\text{pdr}}$
    BLOCK(σ, $i-1$)
  take c minimal s.t. $c \subseteq \neg\sigma_b$ and $\mathcal{H}(\delta, \mathcal{F}_{i-1}^{\text{pdr}}, c) = \mathit{true}$ and $\mathit{Init} \Longrightarrow c$
  for $1 \le j \le i$ **do**
    $\mathcal{F}_j^{\text{pdr}} \leftarrow \mathcal{F}_j^{\text{pdr}} \wedge c$

The Hoare-query model is not specific to PDR. It also captures algorithms in the ICE learning model [GLMN14], as we discuss in §3.5.1, and as result can model algorithms captured by the ICE model (see §2.2.2). In §3.5.2 we show that the Hoare-query model is in fact strictly more powerful than the ICE model.

Remark 3.3.4. *Previous black-box models for invariant inference [GLMN14] encapsulated access also to Init, Bad. In our model we encapsulate only access to δ, since (1) it is technically simpler, (2) a simple transformation can make Init, Bad uniform across all programs, embedding the differences in the transition relation; indeed, our constructions of classes of transition systems in this chapter are such that Init, Bad are the same in all transition systems that share a vocabulary, hence Init, Bad may be inferred from the vocabulary. (Unrestricted access to Init, Bad is stronger, thus lower bounds on our models apply also to models restricting access.)*

Complexity.

Focusing on *information*, we do not impose computational restrictions on the algorithms, and only count the number of queries the algorithm performs to reveal information of the transition relation. In particular, when establishing lower bounds on the query complexity, we even consider algorithms that may compute non-computable functions. However, whenever we construct algorithms demonstrating upper bounds on query complexity, these algorithms in fact have polynomial *time* complexity, and we note this when relevant.

Given a query oracle and an inference algorithm that uses it, we analyze the number of queries the algorithm performs as a function of $n = |\Sigma|$, in a *worst-case* model w.r.t. to possible transition systems in the class of interest (over Σ).

The definition is slightly complicated by considering, as we do later in the chapter, query-models in which more than one oracle exists, i.e., an algorithm may use any oracle from a *family* of query oracles. In this case, we analyze the query complexity of an algorithm in a *worst-case* model w.r.t. the possible query oracles in the family as well.

Formally, the query complexity is defined as follows:

Definition 3.3.5 (Query Complexity)**.** *For a class of transitions systems $\mathcal{P}$, the* query complexity *of (a possibly computationally unrestricted) $\mathcal{A}$ w.r.t. a query oracle family $\mathcal{Q}$ is defined as*

$$q_{\mathcal{A}}^{\mathcal{Q}}(n) = \sup_{Q \in \mathcal{Q}} \sup_{\substack{(Init,\delta,Bad) \in \mathcal{P}, \\ |\Sigma|=n}} \#query(\mathcal{A}^{Q}(Init, Bad, [\delta])) \tag{3.1}$$

where $\#query(\mathcal{A}^{Q}(Init, Bad, [\delta]))$ is the number of times the algorithm accesses Q given this oracle and the input. (These numbers might be infinite.)

The query complexity in the Hoare-query model is $q_{\mathcal{A}}^{\{\mathcal{H}\}}(n)$.

Remark 3.3.6. *In our definition, query complexity is a function of the size of the vocabulary $n = |\Sigma|$, but not of the size of the representation of the transition relation $|\delta|$. This reflects the fact that an algorithm in the black-box model does not access δ directly. In the open questions listed in Chapter 6 we discuss taking $|\delta|$ as an additional complexity parameter. The drawback is that learning δ itself becomes feasible, undermining the black-box model. Efficiently learning δ is possible when using unlimited computational power and exponentially-long queries. However, whether the same holds when using unlimited computational power with only polynomially-long queries is related to open problems in classical concept learning.*

3.4 The Information Complexity of Hoare-Query Algorithms

In this section we prove an information-based lower bound on Hoare-query invariant inference algorithms, and also extend the results to algorithms using *interpolation*, another SAT-based operation. We then apply these results to study the role of information in generalization as part of inference algorithms.

3.4.1 Information Lower Bound for Hoare-Query Inference

We show that a Hoare-query inference algorithm requires $2^{\Omega(n)}$ Hoare-queries in the worst case to decide whether a CNF invariant of length polynomial in n exists. (Recall that n is a shorthand for $|\Sigma|$, the size of the vocabulary of the input transition system.) This result applies even when allowing the choice of queries to be inefficient, and when allowing the queries to use exponentially-long formulas. It provides a lower bound on the time complexity of actual algorithms, such as PDR, that are captured by the model. Formally:

Theorem 3.4.1. *Every Hoare-query inference algorithm $\mathcal{A}^{\mathcal{H}}$ deciding polynomial-length inference for the class of all propositional transition systems has query complexity of $2^{\Omega(n)}$.*

The rest of this section proves a strengthening of this theorem, for a specific class of transition systems (which we construct next), for any class of invariants that includes antimonotone CNF (Def. 2.3.7), and for computationally-unrestricted algorithms:

Theorem 3.4.2. *Every Hoare-query inference algorithm $\mathcal{A}^{\mathcal{H}}$, even computationally-unrestricted, deciding invariant inference for the class of transition systems $\mathcal{P}_{\Sigma_2^P}$ (§3.4.1) and for any class of target invariants $\mathcal{L}$ s.t.* Mon-CNF$_n \subseteq \mathcal{L}$. *has query complexity of $2^{\Omega(n)}$.*

(That classes containing Mon-CNF$_n$ are already hard becomes important later, in §3.5.)

A Hard Class of Transition Systems

In this section we construct a $\mathcal{P}_{\Sigma_2^P}$, a hard class of transition systems, on which we prove hardness results.

The QBF$_2$ problem The construction of $\mathcal{P}_{\Sigma_2^P}$ is based on the prototypical Σ_2^P-complete problem of QBF$_2$ from classical computational complexity theory. In essence, this is a propositional satisfiability problem, except that instead of checking whether there *exists* a satisfying assignment as in the classical satisfiability problem, we are interested whether there is an *exists-forall* assignment. Formally, in this problem, the input is a quantified Boolean formula $\exists \overline{y}. \forall \overline{x}.\ \phi(\overline{x}, \overline{y})$ where ϕ is a Boolean (quantifier-free) formula, and the problem of QBF$_2$ is to decide whether the quantified formula is *true*, namely, whether there exists a Boolean assignment to $\overline{y}$ s.t. $\phi(\overline{x}, \overline{y})$ is true for every Boolean assignment to $\overline{x}$.

The class $\mathcal{P}_{\Sigma_2^P}$. We shall now use the QBF$_2$ problem to construct a set of transition systems. The idea is that each transition system in the class is a program that solves the QBF$_2$ problem for a specific formula, by iterating through the solutions one by one. Invariant inference for such systems boils down to solving the same problem, which is comptuationally hard, and, as we shall see, also hard in an information-theoretical way. Let us begin:

For each $k \in \mathbb{N}$, we define $\mathcal{P}^k_{\Sigma_2^P}$, and finally take $\mathcal{P}_{\Sigma_2^P} = \bigcup_{k\in\mathbb{N}} \mathcal{P}^k_{\Sigma_2^P}$.

Let $k \in \mathbb{N}$. For each formula $\exists \overline{y}. \forall \overline{x}. \phi(\overline{y}, \overline{x})$, where $\overline{y} = y_1, \ldots, y_k$, $\overline{x} = x_1, \ldots, x_k$ are variables and ϕ is a quantifier-free formula over the variables $\overline{x} \cup \overline{y}$, we define a transition system $TS^{\phi} = (Init_k, \delta^P_{\phi}, Bad_k)$. Intuitively, it iterates through $\overline{y}$ lexicographically, and for each $\overline{y}$ it iterates lexicographically through $\overline{x}$ and checks if all assignments to $\overline{x}$ satisfy $\phi(\overline{y}, \overline{x})$. If no such $\overline{y}$ is found, this is an error. More precisely,

1. $\Sigma_k = \{y_1, \ldots, y_k, x_1, \ldots, x_k, a, b, e\}$.
2. $Init_k = \overline{y} = \overline{0} \wedge \overline{x} = \overline{0} \wedge \neg a \wedge b \wedge \neg e$.
3. $Bad_k = e$.
4. δ^P_{ϕ}: evaluate $\phi(\overline{y}, \overline{x})$, and perform the following changes (at a single step): If the result is false, set a to true. If $\overline{x} = \overline{1}$ and a is still false, set b to false. If in the pre-state $\overline{x} = \overline{1}$, increment $\overline{y}$ lexicographically, reset a to false, and set $\overline{x} = 0$; otherwise increment $\overline{x}$ lexicographically. If in the pre-state $\overline{y} = \overline{1}$, set e to b. (Intuitively, a is false as long as no

falsifying assignment to $\overline{x}$ has been encountered for the current $\overline{y}$, b is true as long as we have not yet encountered a $\overline{y}$ for which there is no falsifying assignment.)

We denote the resulting class of transition systems $\mathcal{P}^k_{\Sigma^P_2} = \{ TS^\phi \mid \phi = \phi(y_1, \ldots, y_k, x_1, \ldots, x_k)\}$.

The following lemma relates the QBF_2 problem for ϕ to the inference problem of TS^ϕ:

Lemma 3.4.3. *Let $TS^\phi \in \mathcal{P}^k_{\Sigma^P_2}$. Then TS^ϕ is safe iff it has an inductive invariant in* Mon-CNF_{2k+1} *iff the formula $\exists \overline{y}. \forall \overline{x}. \phi(\overline{y}, \overline{x})$ is true.*

Proof. There are two cases:

- If $\exists \overline{y}. \forall \overline{x}. \phi(\overline{y}, \overline{x})$ is true, let $\overline{v}$ be the first valuation for $\overline{y}$ that realizes the existential quantifiers. Then the following is an inductive invariant for TS^ϕ:

$$I = \neg e \wedge (b \rightarrow \overline{y} \leq \overline{v}) \wedge ((b \wedge a) \rightarrow \overline{y} < \overline{v}) \tag{3.2}$$

  where the lexicographic constraint is expressed by the following recursive definition on $\overline{y}_{[d]} = (y_1, \ldots, y_d), \overline{v}_{[d]} = (v_1, \ldots, v_d)$:

$$\overline{y}_{[d]} < \overline{v}_{[d]} \stackrel{\text{def}}{=} \begin{cases} \neg y_d \vee (\overline{y}_{[d-1]} < \overline{v}_{[d-1]}) & v_d = \mathit{true} \\ \neg y_d \wedge (\overline{y}_{[d-1]} < \overline{v}_{[d-1]}) & v_d = \mathit{false} \end{cases}$$

  and $\overline{y} \leq \overline{v} \stackrel{\triangle}{=} \overline{y} < (\overline{v} + 1)$ (or *true* if $\overline{v} = \overline{1}$).

  $I \in$ Mon-CNF_{2k+1}: Note that $\overline{y}_{[k]} < \overline{v}_{[k]}$ can be written in CNF with at most n clauses: in the first case a literal is added to each clause, and in the second another clause is added. Thus I can be written in CNF with at most $2k + 1$ clauses. Further, the literals of $\overline{y}$ appear only negatively in $\overline{y}_{[k]} < \overline{v}_{[k]}$, and hence also in I. The other literals $(\neg e, \neg a, \neg b)$ also appear only negatively in I. Hence, I is antimonotone.

  I is indeed an inductive invariant: it is straightforward that I includes the initial states and excluded the bad states. To see that it is inductive, consider a valuation to $\overline{y}$ in a pre-state satisfying the invariant. (We abuse notation and refer to the valuation by $\overline{y}$.) There are three cases:

  - If $\overline{y} < \overline{v}$, then (i) $\neg e$ is retained by a step, (ii) $\overline{y} \leq \overline{v}$ holds after a step, (iii) $\overline{y} < \overline{v}$ still holds unless the transition is from the last evaluation for $\overline{y} = \overline{v} - 1$ to $\overline{v}$, in which case a is turned to *false*.
  - If $\overline{y} > \overline{v}$, the invariant guarantees that in the pre-state b is false, and thus e remains false after a step. b also remains false and thus the rest of the invariant also holds in the post-state.
  - If $\overline{y} = \overline{v}$, the invariant guarantees that in the pre-state either b is false or a is false. If b is false the same reasoning of the previous case applies. Otherwise, we have that

a is false. By the definition of $\overline{v}$ all valuations for $\overline{x}$ results in $\phi(\overline{v}, \overline{x}) = \mathit{true}$, so a remains false after a step, and once we finish iterating through $\overline{x}$ we set b to false immediately.

The claim follows.

- If $\exists \overline{y}. \forall \overline{x}. \phi(\overline{y}, \overline{x})$ is not true, then TS^{ϕ} is not safe (and thus does not have an inductive invariant of any length). This is because for every valuation of $\overline{y}$ a violating $\overline{x}$ is found, turning a to true, and b never turns to false, so after iterating through all possible $\overline{y}$'s e will become true.

□

Before we turn to prove Thm. 3.4.2 and establish a lower bound on the query complexity in the Hoare model, we note that this construction also yields the computational hardness mentioned in §3.2:

Proof of Thm. 3.2.3. The upper bound is straightforward: guess an invariant in $\mathrm{CNF}_{p(n)}$ and check it. For the lower bound, use the reduction outlined above: given $\phi(y_1, \ldots, y_k, x_1, \ldots, x_k)$, construct TS^{ϕ}. Note that the vocabulary size, n, is $2k + 3$, and the invariant, when exists, is of length at most $2k + 1 \leq n$.[7] The reduction is polynomial as the construction of TS^{ϕ} (and n) from ϕ is polynomial in k and $|\phi|$: note that lexicographic incrementation can be performed with a propositional formula of polynomial size. □

Lower Bound's Proof

We now turn to prove Thm. 3.4.2. Given an algorithm with polynomial query complexity, the proof constructs two transition system: one that has a polynomial-length invariant and one that does not, and yet all the queries the algorithm performs do not distinguish between them. The construction uses the path the algorithm takes when all Hoare queries return *false* as much as possible. Intuitively, such responses are less informative and rule out less transition relations, because they merely indicate the existence of a single counterexample to a Hoare triple, as opposed to the result *true* which indicates that *all* transitions satisfy a property.

Proof of Thm. 3.4.2. Let $\mathcal{A}$ be a computationally unbounded Hoare-query algorithm. We show that the number of Hoare queries performed by $\mathcal{A}$ on transition systems from $\mathcal{P}_{\Sigma_2^P}$ with $|\Sigma| = n$ is at least $2^{\frac{n-1}{2}}$. To this end, we show that if $\mathcal{A}$ over $|\Sigma| = 2n + 3$ performs less than 2^n queries, then there exist two formulas ψ_1, ψ_2 over $y_1, \ldots, y_n, x_1, \ldots, x_n$ such that

- all the Hoare queries performed by $\mathcal{A}$ on $\delta^P_{\psi_1}$ and $\delta^P_{\psi_2}$ (the transition relations of TS^{ψ_1} and TS^{ψ_2}, respectively) return the same result, even though

[7] For an arbitrary polynomial $p(n) = \Omega(n)$, e.g., $p(n) = c \cdot n$ with $0 < c < 1$, enlarge Σ, e.g., by adding to *Init* initialization of fresh variables that are not used elsewhere, to ensure existence of an invariant of length $\leq p(n)$.

- $\mathcal{A}$ should return different results when run on $TS^{\psi_1} \in \mathcal{P}^n_{\Sigma^P_2}$ and $TS^{\psi_2} \in \mathcal{P}^n_{\Sigma^P_2}$, since TS^{ψ_1} has an invariant in Mon-CNF$_{2n+1}$ and TS^{ψ_2} does not have an invariant (of any length).

We begin with some notation. Running on input TS^{ϕ}, we abbreviate $\mathcal{H}(\delta^P_\phi, \cdot, \cdot)$ by $\mathcal{H}(\phi, \cdot, \cdot)$. Denote the queries $\mathcal{A}$ performs and their results by $\mathcal{H}(\phi, \alpha_1, \beta_1) = b_1, \ldots, \mathcal{H}(\phi, \alpha_m, \beta_m) = b_m$. We call an index i *sat* if $b_i = \mathit{false}$. We say that ψ query-agrees with ϕ if $\mathcal{H}(\psi, \alpha_i, \beta_i) = b_i$ for all i. We say that ψ sat-query-agrees with ϕ if for every i such that $b_i = \mathit{false}$ it holds that $\mathcal{H}(\psi, \alpha_i, \beta_i) = \mathit{false}$.

We first find a formula ϕ over $y_1, \ldots, y_n, x_1, \ldots, x_n$ such that the sequence of queries $\mathcal{A}$ performs when executing on TS^{ϕ} is *maximally satisfiable*: if ψ sat-query-agrees with ϕ, then ψ (completely) query-agrees with ϕ on the queries, that is,

$$\forall \psi. \quad (\forall i.\ b_i = \mathit{false} \Rightarrow \mathcal{H}(\psi, \alpha_i, \beta_i) = b_i) \Longrightarrow (\forall i.\ \mathcal{H}(\psi, \alpha_i, \beta_i) = b_i) \tag{3.3}$$

We construct this sequence iteratively (and define ϕ accordingly) by always taking ϕ so that the result of the next query is *false* as long as this is possible while remaining consistent with the results of the previous queries: Initially, choose some arbitrary ϕ^0. At each point i, consider the first i queries $\mathcal{A}$ performs on ϕ^i, $\mathcal{H}(\phi^i, \alpha_1, \beta_1) = b_1, \ldots, \mathcal{H}(\phi^i, \alpha_i, \beta_i) = b_i$. If $\mathcal{A}$ terminates without performing another query, we are done: the desired ϕ is ϕ_i. Otherwise let $(\alpha_{i+1}, \beta_{i+1})$ be the next query. Amongst formulas ϕ^{i+1} that query-agree on the first i queries, namely, $\mathcal{H}(\phi^{i+1}, \alpha_j, \beta_j) = b_j$ for all $j \leq i$, choose one such that $\mathcal{H}(\phi^{i+1}, \alpha_{i+1}, \beta_{i+1}) = \mathit{false}$ if possible; if such ϕ^{i+1} does not exist take e.g. $\phi^{i+1} = \phi^i$. The dependency of $\mathcal{A}$ on ϕ^i is solely through the results of the queries to $\mathcal{H}(\delta^P_\phi, \cdot, \cdot)$, so $\mathcal{A}$ performs the same i initial queries when given ϕ^{i+1}. The result is a maximally satisfiable sequence, for if a formula ψ differs in query $i+1$ in which the result is *false* instead of *true* we would have taken such a ψ as ϕ^{i+1}.

Let ϕ be such a formula with a maximally satisfiable sequence of queries $\mathcal{A}$ performs on ϕ, $\mathcal{H}(\phi, \alpha_1, \beta_1) = b_1, \ldots, \mathcal{H}(\phi, \alpha_m, \beta_m) = b_m$. For every sat i, take a counterexample $\sigma_i, \sigma'_i \models \alpha_i \wedge \delta^P_\phi \wedge \neg\beta'_i$. The single transition (σ_i, σ'_i) of δ^P_ϕ depends on the value of ϕ on at most one assignment to $\overline{x}, \overline{y}$, so there exists a valuation $v_i : \overline{y} \cup \overline{x} \to \{\mathit{true}, \mathit{false}\}$ such that

$$\forall \psi.\ \ \psi(v_i) = \phi(v_i) \Longrightarrow \sigma_i, \sigma'_i \models \alpha_i \wedge \delta^P_\psi \wedge \neg\beta'_i \tag{3.4}$$

as well. It follows that

$$\forall \psi.\ \ \psi(v_i) = \phi(v_i) \Longrightarrow \mathcal{H}(\psi, \alpha_i, \beta_i) = \mathcal{H}(\phi, \alpha_i, \beta_i) = \mathit{false}. \tag{3.5}$$

Let $v_{i_1}, \ldots, v_{i_t}$ be the valuations derived from the sat queries (concerning indexing, $b_i = \mathit{false}$ iff $b_i = b_{i_j}$ for some j). We say that a formula ψ *valuation-agrees* with ϕ on $v_{i_1}, \ldots, v_{i_t}$ if $\psi(v_{i_j}) = \phi(v_{i_j})$ *for all* j's. Since the sequence of queries is maximally satisfiable, if ψ valuation-agrees with ϕ on $v_{i_1}, \ldots, v_{i_t}$ then ψ query-agrees with ϕ, namely, $\mathcal{H}(\psi, \alpha_i, \beta_i) = \mathcal{H}(\phi, \alpha_i, \beta_i)$ for *all* $i = 1, \ldots, m$. As the dependency of $\mathcal{A}$ on ϕ is solely through the results $b_1, \ldots, b_m$, it follows

that $\mathcal{A}$ performs the same queries on ψ as it does on ϕ and returns the same answer.

It remains to argue that if $m < 2^n$ then there exist two formulas ψ_1, ψ_2 that valuation-agree with ϕ on $v_{i_1}, \ldots, v_{i_t}$ but differ in the correct result $\mathcal{A}$ should return: $\exists \overline{y}. \forall \overline{x}. \psi_1(\overline{y}, \overline{x})$ is true, and so TS^{ψ_1} has an invariant in Mon-CNF$_{2n+1}$ (Lemma 3.4.3), whereas $\exists \overline{y}. \forall \overline{x}. \psi_2(\overline{y}, \overline{x})$ is not, and so TS^{ψ_2} does not have an invariant of any length or form (Lemma 3.4.3). This is possible because the number of constraints imposed by valuation-agreeing with ϕ on $v_{i_1}, \ldots, v_{i_t}$ is less than the number of possible valuations of $\overline{x}$ for every valuation of $\overline{y}$ and vice versa:

$$\psi_1(\overline{y}, \overline{x}) = \bigwedge_{\substack{i=1..t \\ \theta(v_{i_j})=false}} (\overline{y}, \overline{x}) \neq v_{i_j} \tag{3.6}$$

is true on all valuations except for some of $v_{i_1}, \ldots, v_{i_t}$, and since $t \leq m < 2^n$ there exists some $\overline{y}$ such that for all $\overline{x}$, $(\overline{y}, \overline{x})$ is not one of these valuations (recall that $|y| = n$ bits). Dually,

$$\psi_2(\overline{y}, \overline{x}) = \bigvee_{\substack{i=1..t \\ \theta(v_{i_j})=true}} (\overline{y}, \overline{x}) = v_{i_j} \tag{3.7}$$

is false on all valuations except for some of $v_{i_1}, \ldots, v_{i_t}$, and since $t \leq m < 2^n$ for every $\overline{y}$ there exists $\overline{x}$ such that $(\overline{y}, \overline{x})$ is not one of these valuations (recall that $|y| = n$ bits). This concludes the proof. □

Remark 3.4.4 (Extension to Ternary Simulation)**.** *The lower bound of Thm. 3.4.2 continues to hold when the algorithm is equipped with a query oracle that can implement ternary simulation [EMB11]: for a transition relation δ and input formulas $\alpha, \beta \in \text{wff}(\Sigma)$, the* Proof obligation extension oracle*, $\mathcal{T}(\alpha, \beta)$, returns true if $\forall \sigma \in \alpha. \exists \sigma' \in \beta. (\sigma, \sigma') \in \delta$; otherwise it returns false. Intuitively, $\mathcal{T}(\alpha, \beta) = true$ means that if β must be excluded from the invariant (e.g., because all the states in β reach Bad), then so must α.*

The proof argument for Thm. 3.4.2 described in this section can be extended to algorithms that use $\mathcal{H}$ as well as $\mathcal{T}$ in the same way. The reason is that $\mathcal{T}(\alpha, \beta) = false$ follows from the existence of a single transition $(\sigma, \sigma') \in \delta$ s.t. $\sigma \models \alpha, \sigma' \not\models \beta$ in the transition systems in $\mathcal{P}_{\Sigma_2^P}$ (which are deterministic). Hence, choosing such a transition and including it in both TS^{ψ_1}, TS^{ψ_2} suffices for them to agree on $\mathcal{T}(\alpha, \beta) = false$, and the rest of the proof is without change.

3.4.2 Extension to Interpolation-Based Algorithms

We now consider inference algorithms based on *interpolation*, another operation supported by SAT solvers. Interpolation has been introduced to invariant inference by McMillan [McM03], and extended in many works since (see §2.2.2).

Interpolation algorithms infer invariants from facts obtained with Bounded Model Checking (BMC), which we formalize as follows:

Definition 3.4.5 (Bounded Reachabilitiy Check)**.** *The* k-bounded reachability check *returns*

$$\mathcal{H}^{(k)}(\delta, \alpha, \beta) \stackrel{\text{def}}{=} \alpha(\Sigma^0) \wedge \delta(\Sigma^0, \Sigma^1) \wedge \ldots \wedge \delta(\Sigma^{k-1}, \Sigma^k) \implies \beta(\Sigma^k) \tag{3.8}$$

for $\alpha, \beta \in \text{wff}(\Sigma)$*, where* $\Sigma^0, \ldots, \Sigma^k$ *are* $k+1$ *distinct copies of the vocabulary.*

An inference from queries algorithm that uses $\mathcal{H}^{(k)}(\cdot,\cdot,\cdot)$ is called an *extended Hoare-query algorithm*. Such algorithms are studied in depth in Chapter 4. Here, we use BMC to define the interpolation-query oracles:

Definition 3.4.6 (Interpolation-Query Model)**.** *An* interpolation-query oracle *is a query oracle* Q *such that for every* δ*,* $\alpha, \beta \in \text{wff}(\Sigma)$*, and* $k_1, k_2 \in \mathbb{N}$*,*

- $Q^{(k_1,k_2)}(\delta, \alpha, \beta) = \bot$ *if* $\mathcal{H}^{(k_1+k_2)}(\delta, \alpha, \beta) =$ *false, and*
- $Q^{(k_1,k_2)}(\delta, \alpha, \beta) = \rho$ *for* $\rho \in \text{wff}(\Sigma)$ *such that* $\mathcal{H}^{(k_1)}(\delta, \alpha, \rho) =$ *true and* $\mathcal{H}^{(k_2)}(\delta, \rho, \beta) =$ *true otherwise.*

We define ITP *to be the* family *of all interpolation-query oracles.*

An algorithm in the interpolation-query model*, also called an* interpolation-query algorithm*, is an inference from queries algorithm expecting any interpolation query oracle, where* k_1, k_2 *are bounded by a* polynomial *in* n *in all queries. The query complexity in this model is* $q^{ITP}_{\mathcal{A}}(n)$*.*

Interpolation-query oracles form a *family* of oracles since different oracles can choose different ρ for every $\delta, \alpha, \beta, k_1, k_2$. Note that ρ may be exponentially long.

Lower Bound on Interpolation-Query Algorithms

We show an exponential lower bound on query complexity for interpolation-query algorithms. To this end we prove the following adaptation of Thm. 3.4.2:

Theorem 3.4.7. *Every interpolation-query inference algorithm, even computationally-unrestricted, deciding polynomial-length inference for the class of transition systems* $\mathcal{P}_{\Sigma_2^P}$ *(§3.4.1) has query complexity of* $2^{\Omega(n)}$*.*

We remark that the lower bound on the interpolation-query model does not follow directly from the result for the Hoare-query model: an interpolant for $\mathcal{H}^{(k_1+k_2)}(\delta, \alpha, \beta) =$ *true* depends on all traces of length $k_1 + k_2$ starting from states satisfying α, which may be an exponential number, so it cannot be computed simply by performing a polynomial number of Hoare queries to find these traces and computing an interpolant based on them. In principle, then, an interpolant can convey information beyond a polynomial number of Hoare queries. Our proof argument is therefore more subtle: we show that there exists a choice of an interpolant that is not more informative than the existence of some interpolant (i.e., only reveals information on $\mathcal{H}^{(k_1+k_2)}(\cdot,\cdot,\cdot)$), in the special case of systems in $\mathcal{P}_{\Sigma_2^P}$, in the maximally satisfiable branch of an algorithm's execution as used in the proof of Thm. 3.4.2.

Proof. Let $\mathcal{A}$ be a computationally unbounded algorithm using bounded reachability queries with bounds $k_1, k_2 < r(n)$ for some polynomial $r(n)$ (here $n = (|\Sigma| - 1)/2$, as in the proof of Thm. 3.4.2), with query complexity $m < \frac{2^n}{r(n)}$.

To prove the theorem it is convenient to first consider the algorithmic model which performs bounded-reachability queries of polynomial depth but not interpolation queries, intuitively performing BMC but without obtaining interpolants from the SAT solver. Formally we consider Def. 3.4.5 as a query-oracle and first prove the lower bound for algorithms using this query oracle. The proof follows the argument from the proof of Thm. 3.4.2, relying on the fact that the BMC bounds $k_1, k_2 < r(n)$

Assume that $\mathcal{A}$ performs only bounded reachability queries (without obtaining interpolants). In what follows, we abbreviate $\mathcal{H}^{(k)}(\delta^P_\phi, \cdot, \cdot)$ by $\mathcal{H}^{(k)}(\phi, \cdot, \cdot)$. We start the same as the proof of Thm. 3.4.2 to obtain a formula ϕ such that the sequence of bounded reachability queries $\mathcal{A}$ performs when executing on TS^ϕ is maximally satisfiable. In this proof this reads, $\forall \psi.\ \left(\forall i.\ b_i = \mathit{false} \Rightarrow \mathcal{H}^{(k_i)}(\phi, \alpha_i, \beta_i) = b_i\right) \Longrightarrow \left(\forall i.\ \mathcal{H}^{(k_i)}(\phi, \alpha_i, \beta_i) = b_i\right)$.

The main difference is that every sat query produces a counterexample *trace* rather than a counterexample *transition* as in Thm. 3.4.2. For every sat query i, we take a counterexample trace $\sigma^i_1, \ldots, \sigma^i_{k_i}$, namely, $\sigma^i_1 \models \alpha, \sigma^i_{k_i} \models \neg\beta$, and $\sigma^i_j, \sigma^i_{j+1} \models \delta^P_\phi$. Every such transition $\sigma^i_j, \sigma^i_{j+1} \models \delta^P_\phi$ depends on at most one valuation of ϕ. Thus there exists valuations $v^i_1, \ldots, v^i_{k_i}$ so that every ψ that valuation-agrees with ϕ on these valuations also allows the aforementioned counterexample trace, and thus $\mathcal{H}^{(k_i)}(\psi, \alpha_i, \beta_i) = \mathit{false}$ as well. As in the proof of Thm. 3.4.2, it follows that if ψ valuation-agrees with ϕ on all valuations $v^1_1, \ldots, v^1_{k_1}, \ldots, v^m_1, \ldots, v^m_{k_m}$, then all queries on ψ give that same result as those on ϕ. Since $k_1, \ldots, k_m < r(n)$, the number of these valuations is less that $r(n) \cdot m < 2^n$. The rest of the proof is exactly as in Thm. 3.4.2, constructing two formulas ψ_1, ψ_2 valuation-agreeing with ϕ on all valuations, but one is true (in the QBF_2 sense) and the other is not.

We now turn to interpolation queries. Assume without loss of generality that every interpolation query is preceded by a bounded reachability query with the same bound, and that if the result is *false* the algorithm skips the interpolation query.

Consider the algorithm's execution on ϕ constructed above. We show that there exists an interpolation-query oracle that returns the same interpolants on the queries performed by the algorithm for both ψ_1, ψ_2 from above, and thus the algorithm's execution still does not distinguish between them.

Consider a point when the algorithms seeks an interpolant based on $\mathcal{H}^{(k_1+k_2)}(\phi, \alpha, \beta) = \mathit{true}$. Let S be the set of all formulas consistent with ϕ on all the valuations v_i from above. In particular, $\mathcal{H}^{(k_1+k_2)}(\theta, \alpha, \beta) = \mathit{true}$ for all $\theta \in S$. We construct a single interpolant valid for all $\theta \in S$, that is, a formula ρ s.t. for every $\theta \in S$, $\mathcal{H}^{(k_1)}(\theta, \alpha, \rho) = \mathit{true}$ and $\mathcal{H}^{(k_2)}(\theta, \rho, \beta) = \mathit{true}$. In particular, $\psi_1, \psi_2 \in S$, so this gives the desired interpolant for them.

Take $\hat{\delta} = \bigvee_{\theta \in S} \delta^P_\theta$. We argue that $\mathcal{H}^{(k_1+k_2)}(\hat{\delta}, \alpha, \beta) = \mathit{true}$. For otherwise, there exists a trace $(\sigma_0, \ldots, \sigma_{k_1+k_2})$ of $\hat{\delta}$ such with $\sigma_0 \models \alpha$ and $\sigma_{k_1+k_2} \not\models \beta$. By the definition of $\hat{\delta}$, each

transition (σ_j, σ_{j+1}) originates from $\delta^P_{\theta_j}$ for some $\theta_j \in S$. As before, this transition depends on the truth value of θ_j at one valuation $\hat{v}_j$ at the most. Furthermore, $\hat{v}_1, \ldots, \hat{v}_{k_1+k_2-1}$ are successive valuations, because all the transition systems in the class increment the valuation in the same way. Thus $\hat{v}_{j_1} \neq \hat{v}_{j_2}$. Take a formula $\hat{\theta} \in S$ that agrees with θ_j on $\hat{v}_j$ for all $j = 1, \ldots, k_1 + k_2 - 1$; one exists because

1. $\theta_j(\hat{v}_j)$ cannot contradict the valuations $\phi(v_i)$, because $\theta_i \in S$, and
2. $\theta_{j_1}(\hat{v}_{j_1})$ does not contradict $\theta_{j_2}(\hat{v}_{j_2})$ for $\hat{v}_{j_1} \neq \hat{v}_{j_2}$ for $j_1 \neq j_2$.

Thus $(\sigma_0, \ldots, \sigma_{k_1+k_2})$ is also a valid trace of $\hat{\theta} \in S$, which is a contradiction to $\mathcal{H}^{(k_1+k_2)}(\hat{\theta}, \alpha, \beta) = \mathit{true}$.

Thus there exists some ρ an interpolant for $\hat{\delta}, \alpha, \beta, k_1, k_2$. We choose the interpolation query oracle so that

$$Q^{(k_1,k_2)}(\theta, \alpha, \beta) = \rho$$

for all $\theta \in S$. This is a valid choice of interpolant: $\mathcal{H}^{(k_1)}(\theta, \alpha, \rho) = \mathit{true}$ and $\mathcal{H}^{(k_2)}(\theta, \rho, \beta) = \mathit{true}$ because $\mathcal{H}^{(k_1)}(\hat{\delta}, \alpha, \rho) = \mathit{true}$ and $\mathcal{H}^{(k_2)}(\hat{\delta}, \rho, \beta) = \mathit{true}$ and $\hat{\delta}$ includes all the transitions of δ^P_θ.

The claim follows. □

3.4.3 Impossibility of Generalization from Partial Information

Algorithms such as PDR use generalization schemes to generalize from specific states to clauses (see §3.1.2 and §2.2.2). It is folklore that "good" generalization is the key to successful invariant inference. In this section, we apply the results of §3.4.1 to shed light on the question of generalization. Technically, this is a discussion of the results in §3.4.1.

Clearly, if the generalization procedure has *full information*, that is, has unrestricted access to the input—including the transition relation—then unrestricted computational power makes the problem of generalization trivial (as is every other problem!). For example, "efficient" inference can be achieved by a backward reachability algorithm (see §3.1.1) that blocks counterexamples through a generalization that uses clauses from a target invariant it can compute. This setting of full-information, computationally-unrestricted generalization was used by Padon et al. [PMP+16] in an interactive invariant inference scenario.

Our analysis in §3.4.1 implies that the situation is drastically different when generalization possesses *partial information*: the algorithm does not know the transition relation exactly, and only knows the results of a polynomial number of Hoare queries. By Thm. 3.4.2, no choice of generalization made on the basis on this information can in general achieve inference in a polynomial number of steps. This impossibility holds even when generalization uses unrestricted computational power, and thus it is a problem of *information*. To further illustrate the idea of partial information, we note that the problem remains hard even when generalization is equipped with information beyond the results of a polynomial number of Hoare queries, information of the reachability of the transition system from *Init* and backwards from *Bad* in a *polynomial* number

of steps[8]; in contrast, information of the states reachable in *any* number of steps constitutes full information and the problem is again trivial with unrestricted computational power.

Finally, the same challenge of partial information is present in algorithms basing generalization on a polynomial number of interpolation queries, as follows from Thm. 3.4.7.

3.5 The Power of Hoare-Queries

Hoare queries are rich in the sense that the algorithm can choose a precondition α and post-condition β and check $\mathcal{H}(\delta, \alpha, \beta)$, where α may be different from β. As such, algorithms in the Hoare-query model can utilize more flexible queries beyond querying for whether a candidate is inductive. In practice, this richer form of queries facilitates an incremental construction of invariants in complex syntactic forms. For example, PDR [Bra11, EMB11] incrementally learns clauses in different frames via relative inductiveness checks, and interpolation learns at each iteration a term of the invariant from an interpolant [McM03] (see §2.2.2). In this section we analyze this important aspect of the Hoare-query model and show that it can be strictly stronger than inference based solely on presenting whole candidate inductive invariants. We formalize the latter approach by the model of *inductiveness-query algorithms*, closely related to ICE learning [GLMN14], and construct a class of transition systems for which a simple Hoare-query algorithm can infer invariants in polynomial time, but every inductiveness-query algorithm requires an exponential number of queries.

3.5.1 Inductiveness-Query Algorithms

We define a more restricted model of invariant inference using only inductiveness queries.

Definition 3.5.1 (Inductiveness-Query Model)**.** *An* inductiveness-query oracle *is a query oracle Q such that for every δ and $\alpha \in \mathrm{wff}(\Sigma)$ satisfying $Init \implies \alpha$ and $\alpha \implies \neg Bad$,*

- $Q(\delta, \alpha) = true$ *if* $\alpha \wedge \delta \implies \alpha'$*, and*
- $Q(\delta, \alpha) = (\sigma, \sigma')$ *such that* $(\sigma, \sigma') \models \alpha \wedge \delta \wedge \neg\alpha'$ *otherwise.*

We define $\mathcal{I}$ to be the family *of all inductiveness-query oracles.*

An algorithm in the inductiveness-query model*, also called an* inductiveness-query algorithm*, is an inference from queries algorithm expecting any inductiveness query oracle. The query complexity in this model is $q^{\mathcal{I}}_{\mathcal{A}}(n)$.*

Inductiveness-query oracles form a *family* of oracles since different oracles can choose different (σ, σ') for every δ, α. Accordingly, the query complexity of inductiveness-query algorithms is measured as a worst-case query complexity over all possible choices of an inductiveness-query oracle in the family.

[8] This can be shown by noting that in $\mathcal{P}_{\Sigma_2^P}$ (used to establish the exponential lower bound) such polynomial-reachability information can be obtained from a polynomial number of Hoare queries, reducing this scenario to the original setting.

ICE learning and inductiveness-queries The inductiveness-query model is closely related to ICE learning [GLMN14], except here the learner is provided with full information on $Init, Bad$ instead of positive and negative examples (and the algorithm refrains from querying on candidates that do not include $Init$ or do not exclude Bad). This model captures several interesting algorithms (see §2.2.2). Our complexity definition in the inductiveness-query model being the worst-case among all possible oracle responses is in line with the analysis of strong convergence in Garg et al. [GLMN14]. Hence, lower bounds on the query complexity in the inductiveness query model imply lower bounds for the strong convergence of ICE learning. We formalize this in the following lemma, using terminology borrowed from Garg et al. [GLMN14] (see §2.2.2):

Lemma 3.5.2. *Let $\mathcal{P}$ be a class of transition systems, and $\mathcal{L}$ a class of candidate invariants. Assume that deciding the existence of an invariant in $\mathcal{L}$, given an instance from $\mathcal{P}$, requires at least r queries in the inductiveness-query model. Then every strongly-convergent ICE-learner for $(\mathcal{P}, \mathcal{L})$ has round complexity at least r.*

Proof. Given a strongly-convergent ICE-learner $\mathcal{A}$ with round-complexity at most r, we construct an inductiveness-query algorithm for deciding $(\mathcal{P}, C)$ in at most r queries, in the following way. Simulate at most r rounds of $\mathcal{A}$, and implement a teacher as follow: When $\mathcal{A}$ produces a candidate $\theta \in C$,

- Check that $Init \implies \theta$, otherwise produce a positive example, a σ s.t. $\sigma \models Init, \sigma \not\models \theta$;
- Check that $\theta \implies \neg Bad$, otherwise produce a negative example, a σ s.t. $\sigma \models Bad, \sigma \models \theta$;
- Perform an inductiveness query for θ. If θ is inductive, we are done—return *true*. Otherwise, the inductiveness-query oracle produces a counterexample—pass it to $\mathcal{A}$.

If r rounds did not produce an inductive invariant, return *false*.

The teacher we implement always extends the learner's sample with an example that actually is an example in the target description (see §2.2.2), and that rules out the current candidate. Thus, if there exists a correct $h \in C$, $\mathcal{A}$ finds one after at most r iterations, and we return *true*. Otherwise, we terminate after at most r with the last candidate not an inductive invariant, and we return *false*. □

Inductiveness queries vs. Hoare queries Inductiveness queries are specific instances of Hoare queries, where the precondition and postcondition are the same. Since Hoare queries can also find a counterexample in a polynomial number of queries (Lemma 3.3.3), inductiveness-query algorithms can be simulated by Hoare-query algorithms. Our results in the rest of this section establish that the converse is not true.

3.5.2 Separating Inductiveness-Queries from Hoare-Queries

In this section we show that the Hoare query model (Def. 3.3.2) is strictly stronger than the inductiveness query model (Def. 3.5.1). We will prove the following main theorem:

Theorem 3.5.3. *There exists a class of systems $\mathcal{M}_E$ for which*

- *polynomial-length invariant inference has* polynomial *query complexity in the* Hoare-query model *(in fact, also polynomial time complexity modulo the query oracle), but*
- *every algorithm in the* inductiveness-query model *requires an* exponential *number of queries.*

The upper bound is proved in Corollary 3.5.8, and the lower bound in Corollary 3.5.11. Building on this theorem's proof, we also obtain the following:

Theorem 3.5.4. *PDR (Alg. 10) cannot be efficiently simulated in the inductiveness-query model: there is no inductiveness-query algorithm $\mathcal{A}$ that solves polynomial-length inference on every $TS = (Init, \delta, Bad)$ with a number of inductiveness queries that has at most polynomial overhead on the number of Hoare queries performed by PDR—query complexity bounded by $p(n) \cdot \#query(PDR^{\mathcal{H}}(Init, Bad, [\delta]))$ where $n = |\Sigma|$, and $p(\cdot)$ is some polynomial.*

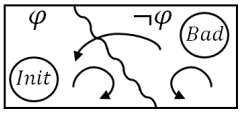
Maximal Transition Systems for Antimonotone Invariants

We first define the transition systems with which we will prove Thm. 3.5.3. The idea is to construct systems that are very simple, in a way that PDR can take advantage of, but ICE algorithms cannot. Specifically, these systems will have the simplest reachability pattern: all reachable states are reachable in one step from the initial states, and all other states reach a bad state in one step. Such a system has exactly one inductive invariant φ—the set of reachable states—and so can be associated with it. Since this system includes all the transitions that can be possibly occur without violating the invariant φ, we call this system the *maximal system w.r.t. φ*. Formally, this is defined as follows:

Definition 3.5.5 (Maximal System)**.** *Let $Init, Bad \not\equiv false$ and let φ be a formula such that $Init \implies \varphi$ and $\varphi \implies \neg Bad$. The* maximal transition system *w.r.t. φ is $(Init, \delta_\varphi^{\mathcal{M}}, Bad)$ where*

$$\delta_\varphi^{\mathcal{M}} = \varphi \to \varphi'.$$

A maximal transition system is illustrated as follows:



Note that $\delta_\varphi^{\mathcal{M}}$ goes from any state satisfying φ to any state satisfying φ, and from any state satisfying $\neg\varphi$ to all states, good or bad. $\delta_\varphi^{\mathcal{M}}$ is *maximal* in the sense that it allows all transitions that do not violate φ being an inductive invariant: any transition relation $\tilde{\delta}$ for which φ is an inductive invariant has $\tilde{\delta} \implies \delta_\varphi^{\mathcal{M}}$.

Lemma 3.5.6. *A maximal transition system $(Init, \delta_\varphi^{\mathcal{M}}, Bad)$ has a unique inductive invariant, φ.*

Proof. Let I be any invariant of $(Init, \delta_\varphi^{\mathcal{M}}, Bad)$. By the definition of $\delta_\varphi^{\mathcal{M}}$ and the fact that $Init \implies \varphi$, the set of states reachable from $Init$ is exactly the set of states satisfying φ. Thus $\varphi \implies I$.

Since $\delta_\varphi^{\mathcal{M}}$ allows transitions from any state satisfying $\neg\varphi$ to Bad, $I \implies \varphi$. □

The class of transition systems on which we focus, $\mathcal{M}_E$, is the class of maximal systems for antimonotone invariants, $\mathcal{M}$, together with certain unsafe systems.

Formally, for each $k \in \mathbb{N}$, we define $\mathcal{M}^k$ as the class of all transition systems $(Init_k, \delta_\varphi^{\mathcal{M}}, Bad_k)$ for $Init_k, Bad_k$ from $\mathcal{P}^k_{\Sigma_2^P}$ (§3.4.1) and $\varphi \in \text{Mon-CNF}_{2k+1}$ such that $Init_k \implies \varphi$ and $\varphi \implies \neg Bad_k$. We then define $\mathcal{M} = \bigcup_{k\in\mathbb{N}} \mathcal{M}^k$. Further, for each k we take the unsafe program $E_k = (Init_k, true, Bad_k)$, and define the class $\mathcal{M}_E = \mathcal{M} \cup \{E_k \mid k \in \mathbb{N}\}$. Below we abbreviate and refer to the class $\mathcal{M}_E$ as "antimonotone maximal systems".

Note that for each k, only a single transition system, E_k, in $\mathcal{M}_E$ does not have an invariant, and the others have an antimonotone invariant. Still, Corollary 3.5.11 establishes a lower bound on polynomial-length inference for $\mathcal{M}_E$ using inductiveness queries. This means that using inductiveness queries alone, it is hard to distinguish between antimonotone invariants (otherwise decision would have been feasible via search). On the other hand, with Hoare queries, search becomes feasible (establishing the upper bound).

Upper Bound for Hoare-Query Algorithms for Antimonotone Maximal Systems

A simple algorithm can find inductive invariants of antimonotone maximal systems with a polynomial number of queries. It is essentially PDR with a single frame. The ability to *find* invariants for $\mathcal{M}_E$ (and check invariants) shows that it is possible to *decide* polynomial-length inference for $\mathcal{M}_E$.

We now present the PDR-1 algorithm (which was also discussed in §3.1.2, and is cast here formally as a Hoare-query algorithm). This is a backward reachability algorithm, operating by repeatedly checking for the existence of a counterexample to induction, and obtaining a concrete example by the method discussed in Lemma 3.3.3. The invariant is then strengthened by conjoining the candidate invariant with the negation of the formula Block returns. This formula is a subset of the cube of the pre-state. In PDR-1, Block performs generalization by dropping a literal from the cube whenever the remaining conjunction does not hold for any state reachable in at most one step from *Init*. The result is the strongest conjunction whose negation does not exclude any state reachable in at most one step. (This might exclude reachable states in general transition systems, but not in antimonotone maximal systems, since maximality ensures that their diameter is one.)

The main property of antimonotone CNF formulas we exploit in the upper bound is the ability to reconstruct them from *prime consequences*, as discussed in §2.3.2. We use this property

Algorithm 11 PDR-1 in the Hoare-query model

```
1:  procedure PDR-1(Init, [δ], Bad)                    // Backward reachability with PDR-1 generalization
2:      I ← ¬Bad
3:      while H(δ, I, I) = false do                    // I not inductive
4:          (σ, σ′) ← MODEL([δ], I, ¬I′)               // counterexample to induction of I. implemented using
    Lemma 3.3.3
5:          d ← BLOCK-PDR-1(Init, [δ], σ)
6:          I ← I ∧ ¬d
        return I
7:
8:  procedure BLOCK-PDR-1(Init, [δ], σ)                // Generalization according to one-step reachability
9:      d ← cube(σ)
10:     for l ∈ cube(σ) do
11:         t ← d \ {l}
12:         if Init ⟹ ¬t ∧ H(δ, Init, ¬t) then        // Init ⟹ t ∧ Init ∧ δ ⟹ ¬t′
13:             d ← t
        return d
```

to show that PDR-1 efficiently finds the invariants of the safe maximal monotone systems $\mathcal{M}$, as implied by the following, slightly more general, lemma:

Lemma 3.5.7. *Let $TS = (Init, \delta, Bad)$ be a transition system over Σ, $n = |\Sigma|$, and $m \in \mathbb{N}$ such that*

1. *TS is safe,*
2. *every reachable state in TS is reachable in at most one step from Init,*
3. *this set can be described by a* Mon-CNF$_m$ *formula, namely, there is $\varphi \in$ Mon-CNF$_m$ such that $\sigma \models \varphi$ iff $\sigma \models Init$ or $\exists \sigma_0 \in Init$ s.t. $(\sigma_0, \sigma) \models \delta$.*

Then PDR-1*$(Init, Bad, [\delta])$ returns the inductive invariant φ for TS with at most $n \cdot m$ Hoare queries.*

Proof. Let φ be as in the premise. We show that I of Alg. 11 (1) always overapproximates φ, (2) is strengthened with a new clause from φ in every iteration.

(1) We claim by induction on the number of iterations in Alg. 11 that $\varphi \implies I$. Clearly this holds initially. In line 4, $\sigma \not\models \varphi$, otherwise $\sigma' \models \varphi$, and $\varphi \implies I$ gives $\sigma' \models I$ which is a contradiction to (σ, σ') being a CTI. Now, for every $\sigma \not\models \varphi$, its minimization $\neg d$ is a consequence of φ, namely, $\varphi \implies \neg d$, because $\neg d$ holds for all states reachable in at most one step ($Init \implies \neg d$, $Init \wedge \delta \implies \neg d'$) and those are states satisfying φ. Thus also $\varphi \implies I \wedge \neg d$.

(2) As we have argued earlier, $\neg d$ in line 5 is a consequence of φ. We refer to it as the clause $c = \neg d$. We argue that c is a clause of φ. By Thm. 2.3.9, is suffices to show that c is a *prime* consequence of φ, seeing that φ is antimonotone. Assume (for the sake of contradiction) that $\tilde{c} \subsetneq c$ is a consequence of φ. Consider the minimization procedure as it attempts to remove a literal $\tilde{l} \in (\neg c) \setminus (\neg \tilde{c})$ (line 11). This literal is not removed, so $Init \wedge t$ or $Init \wedge \delta \wedge t'$ is satisfiable at this point. So there is a state reachable in at most one step that satisfies t, which means that $\varphi \wedge t$ is satisfiable. From this point onwards, literals are only omitted from t, apart from $\tilde{l}$

that is resurrected; thus $\neg c \subseteq t \cup \{\tilde{l}\}$. These are conjunctions, so $\varphi \wedge (\neg c \setminus \{\tilde{l}\})$ is satisfiable. But this means that $\varphi \not\Longrightarrow (c \setminus \{\neg\tilde{l}\})$. In particular, it follows that $\varphi \not\Longrightarrow \tilde{c}$ since $\tilde{c} \subseteq c \setminus \{\neg\tilde{l}\}$. Thus $\tilde{c}$ is not a consequence of φ, which is a contradiction to the premise. Therefore c must be a prime consequence of φ.

It remains to argue that s in line 5 is not already present in I. But this is true because $\sigma \models s$, and $s \not\models I$.

Overall, after at most m such iterations, $I \Longrightarrow \varphi$ from (2), and also $\varphi \Longrightarrow I$ from (1). Thus $I \equiv \varphi$, which is indeed an inductive invariant (it captures exactly the reachable states). Minimization performs $n = |\Sigma|$ queries, and so the total number of queries is at most nm. □

From this lemma and the uniqueness of the invariants (Lemma 3.5.6) the upper bound for $\mathcal{M}_E$ follows easily:

Corollary 3.5.8. *Polynomial-length invariant inference of $\mathcal{M}_E$ can be decided in a polynomial number of Hoare queries.*

Proof. Let $p(\cdot)$ be the polynomial dictating the target length in Def. 3.2.2. The Hoare-query algorithm runs PDR-1 for $p(n) \cdot n$ queries. If PDR-1 does not terminate, return *no*. Otherwise, it produces a candidate invariant ψ. If $\psi \notin \mathrm{CNF}_{p(n)}$, return *no*. Perform another check for whether ψ is indeed inductive: if it is inductive, return *yes*, otherwise *no*.

Correctness: let $(\mathit{Init}, \delta, \mathit{Bad}) \in \mathcal{M}_E$. If there exists an inductive invariant in $\mathrm{CNF}_{p(n)}$, it is unique, and is the formula $\varphi \in \text{Mon-CNF}_{p(n)}$ that characterizes the set of states reachable in at most one step. By Lemma 3.5.7 PDR-1 finds φ in a polynomial number of Hoare queries. Otherwise, $(\mathit{Init}, \delta, \mathit{Bad})$ is not safe, or its unique inductive invariant $\varphi \notin \mathrm{CNF}_{p(n)}$. In both cases PDR-1 cannot produce an inductive invariant in $\mathrm{CNF}_{p(n)}$, and terminates/is prematurely terminated after $p(n) \cdot n$ Hoare queries. □

Remark 3.5.9. *The condition that the invariant is antimonotone in Lemma 3.5.7 can be relaxed to* unate *CNF (Def. 2.3.4): PDR-1 successfully finds an invariant in a polynomial number of Hoare queries also for the class of maximal systems for unate invariants. The proof is the same, using Corollary 2.3.10 instead of Thm. 2.3.9.*

Lower Bound for Inductiveness-Query Algorithms for Antimonotone Maximal Systems

We now prove that every inductiveness-query algorithm for the class of antimonotone maximal systems requires exponential query complexity. The main idea of the proof is that inductiveness-query algorithms are oblivious to adding transitions:

Theorem 3.5.10. *Let X, Y be sets of transition systems, such that Y* covers *the transition relations of X, that is, for every $(\mathit{Init}, \delta, \mathit{Bad}) \in X$ there exists $(\mathit{Init}, \hat{\delta}, \mathit{Bad}) \in Y$ over the same vocabulary s.t.*

1. $\delta \implies \hat{\delta}$, *and*

2. *if* $(Init, \delta, Bad)$ *has an inductive invariant in* $\mathrm{CNF}_{p(n)}$, *then so does* $(Init, \hat{\delta}, Bad)$.

Then if $\mathcal{A}$ *is an inductiveness-query algorithm for* Y *with query complexity* t, *then* $\mathcal{A}$ *is also an inductiveness-query algorithm for* X *with query complexity* t.

Proof. Let $\mathcal{A}$ be an algorithm for Y as in the premise. We show that $\mathcal{A}$ also solves the problem for X. Let $(Init, \delta, Bad) \in X$ and analyze $\mathcal{A}^Q(Init, Bad, [\delta])$, where Q is some inductiveness-query oracle. Consider the first t candidates, $\alpha_1, \ldots, \alpha_t$. If one of them is an inductive invariant for $(Init, \delta, Bad)$, we are done (recall that the inductiveness query is only defined for queries with α_i s.t. $Init \implies \alpha_i$ and $\alpha_i \implies \neg Bad$). If we are not done, let $(Init, \hat{\delta}, Bad) \in Y$ as in the premise for the given $(Init, \delta, Bad)$. We show that in this case $\mathcal{A}^Q(Init, Bad, [\delta])$ simulates $\mathcal{A}^{Q'}(Init, Bad, [\hat{\delta}])$ where Q' is an inductiveness-query oracle derived from Q by $Q'(\hat{\delta}, \alpha_i) = Q(\delta, \alpha_i)$ for all $i = 1, \ldots, t$. Note that $Q'(\hat{\delta}, \cdot)$ is a valid inductiveness-query oracle: by the assumption that α_i is not inductive for δ, $Q(\delta, \alpha) = (\sigma, \sigma')$, that is, $\sigma, \sigma' \models \alpha \wedge \delta \wedge \neg\alpha'$. From condition 1, $\delta \implies \hat{\delta}$, and so we deduce that also $\sigma, \sigma' \models \alpha \wedge \hat{\delta} \wedge \neg\alpha'$. Therefore, after at most t queries, $\mathcal{A}^{Q'}(Init, Bad, [\hat{\delta}])$ terminates, returning either (i) an inductive invariant $\varphi \in \mathrm{CNF}_{p(n)}$ for $(Init, \hat{\delta}, Bad)$, which is also an inductive invariant for $(Init, \delta, Bad)$, by condition 1; or (ii) no inductive invariant in $\mathrm{CNF}_{p(n)}$ for $(Init, \hat{\delta}, Bad)$, in which case this is also true for $(Init, \delta, Bad)$, by condition 2. Either way $\mathcal{A}^Q(Init, Bad, [\delta])$ is correct and uses $\leq t$ queries. □

The lower bound for antimonotone maximal systems results from Thm. 3.5.10 together with the hardness previously obtained in Thm. 3.4.2:

Corollary 3.5.11. *Every inductiveness-query algorithm, even computationally-unrestricted, deciding polynomial-length inference for* $\mathcal{M}_E$ *has query complexity of* $2^{\Omega(n)}$.

Proof. For $\mathcal{P}_{\Sigma_2^P}$ with invariant class Mon-CNF_n, an exponential number of Hoare queries is necessary, by Thm. 3.4.2. It follows that in the inductiveness-query model, an exponential query complexity is also required (since a Hoare-query algorithm can implement a valid inductiveness-query oracle). By arguing that $\mathcal{M}_E$ covers $\mathcal{P}_{\Sigma_2^P}$ we can apply Thm. 3.5.10 to deduce that $\mathcal{M}_E$ with invariant class Mon-CNF_n also necessitates an exponential number of inductiveness queries:

Let $(Init_k, \delta, Bad_k) \in \mathcal{P}_{\Sigma_2^P}$. Recall that in these systems, $n = 2k + 3$ (the vocabulary size). If the system does not have an inductive invariant in Mon-CNF_n, then $E_k = (Init_k, true, Bad_k) \in \mathcal{M}_E$ satisfies the conditions of Thm. 3.5.10 (condition 1 holds as evidently $\delta \implies true$, and condition 2 holds vacuously). Otherwise, there exists an inductive invariant $\varphi \in$ Mon-CNF_n for $(Init_k, \delta, Bad_k)$. In this case, the system $(Init_k, \delta_\varphi^{\mathcal{M}}, Bad_k)$ satisfies the conditions of Thm. 3.5.10: condition 1 is due to the maximality of $\delta_\varphi^{\mathcal{M}}$, and 2 holds as φ is an inductive invariant.

Thus $\mathcal{M}_E$ with the class Mon-CNF_n requires an exponential number of inductiveness queries. Since $\mathcal{M}_E$ has a monotone invariant or none at all, it follows that an exponential number inductiveness queries is also required for $\mathcal{M}_E$ with CNF_n, as desired. □

We note that the transition relations in $\mathcal{M}_E$ are themselves polynomial in $|\Sigma|$. Hence the query complexity in this lower bound is exponential not only in $|\Sigma|$ but also in $|\delta|$ (see Remark 3.3.6).

Finally, it is interesting to notice that the safe systems in $\mathcal{M}_E$ have a *unique* inductive invariant, and still the problem is hard.

PDR Is Not an Inductiveness-Query Algorithm

We now turn to prove Thm. 3.5.4. The results above yield almost that: no inductiveness-query algorithm can simulate PDR-1 on the class $\mathcal{M}_E$ of maximal transitions systems for monotone invariants (with an additional unsafe instance) and PDR-1 is *essentially* PDR with a single frame; it remains to close the gap between PDR-1 and PDR. The differences are not great: in PDR, counterexamples are states that reach a bad state, whereas PDR-1 uses counterexamples to induction;[9] these coincide in maximal systems. Additionally, PDR as written in Alg. 1 opens two or three frames before it finds an invariant in the first frame; the second frame is used to generate counterexamples of depth one, and the third possibly to detect convergence (if no optimizations are employed).

Proof of Thm. 3.5.4. We show that Alg. 10—the formulation of PDR as a Hoare-query algorithm—solves the class $\mathcal{M}_E$ with a polynomial number of Hoare queries, namely, that it also achieves the upper bound of Corollary 3.5.8. The claim follows because no inductiveness-query algorithm can solve the same class with polynomial number of queries (Corollary 3.5.11).

To construct the first frame $\mathcal{F}_1^{\text{pdr}}$, PDR samples states from *Bad* (line 7). If the given transition system is the unsafe instance, then the first state sampled from *Bad* is reachable in one step from *Init*, and this is inferred by the algorithm in line 13. Otherwise, we are analyzing a maximal system for φ. In this case, PDR learns lemmas for $\mathcal{F}_1^{\text{pdr}}$, generalizing the counterexample w.r.t. $\underline{\delta}(\mathit{Init})$ by dropping literals, just like PDR-1; as in Lemma 3.5.7, this generates a prime consequence of φ, and a new one each time, so $\mathcal{F}_1^{\text{pdr}}$ consists of at most m clauses when it excludes *Bad*. At this point PDR opens $\mathcal{F}_2^{\text{pdr}}$ and again starts sampling states *Bad*. If $\mathcal{F}_1^{\text{pdr}} \subseteq \varphi$ then $\underline{\delta}(\mathcal{F}_1^{\text{pdr}}) = \varphi$, and as in the first frame, $\mathcal{F}_2^{\text{pdr}}$ is strengthened with a prime consequence of φ. If $\mathcal{F}_1^{\text{pdr}} \not\subseteq \varphi$, then $\underline{\delta}(\mathcal{F}_1^{\text{pdr}}) = \mathit{true}$, and to block the counterexample in $\mathcal{F}_2^{\text{pdr}}$ the algorithm samples predecessors in $\mathcal{F}_1^{\text{pdr}}$ and generalizes them w.r.t. $\underline{\delta}(\mathit{Init}) = \varphi$—again yielding prime consequences of φ—until no such predecessors exist, which is when we have sufficiently strengthened the first frame so that $\mathcal{F}_1^{\text{pdr}} \subseteq \varphi$. Since always $\varphi = \underline{\delta}(\mathit{Init}) \subseteq \mathcal{F}_1^{\text{pdr}}$, we have at this point that $\mathcal{F}_1^{\text{pdr}} \equiv \varphi$, which is an inductive invariant. When pushing lemmas forward, all the lemmas in $\mathcal{F}_1^{\text{pdr}}$ will be pushed to $\mathcal{F}_2^{\text{pdr}}$, leading to $\mathcal{F}_1^{\text{pdr}} \equiv \mathcal{F}_2^{\text{pdr}}$ and convergence; without this optimization, PDR opens $\mathcal{F}_3^{\text{pdr}}$ and in the same way strengthens $\mathcal{F}_2^{\text{pdr}}$ until also $\mathcal{F}_2^{\text{pdr}} \equiv \varphi$ and converge. In any case, the algorithm uses $O(1)$ frames, each frame consists of at most m clauses, and generating each clause uses $O(n)$ queries, including sampling the counterexample and generalization, a polynomial number of Hoare queries in total. □

[9] PDR-1 (Alg. 11) is more accurately described as dual model-based interpolation-based inference (see Chapter 4) with a reachability bound of $k = 1$.

Table 3.1: Concept vs. invariant learning: query complexity of learning Mon-CNF$_n$

Invariant Inference			Concept Learning	
	Maximal Systems	General Systems		
Inductiveness	Exponential (Corollary 3.5.11)	Exponential (Thm. 3.4.2)	Equivalence	Subexponential[1] / Polynomial[2] [HKSS12, Ang87b]
Hoare	Polynomial (Corollary 3.5.8)	Exponential (Thm. 3.4.2)	Equivalence + Membership	Polynomial [Ang87b]

[1] proper learning
[2] with exponentially long candidates

3.6 Invariant Learning & Concept Learning with Queries

What are the connections and differences between concept learning *formulas* in $\mathcal{L}$ and learning *invariants* in $\mathcal{L}$? Can concept learning algorithms be translated to inference algorithms? These questions have spurred much research [e.g. GLMN14, JS17]. In this section we study these questions with the tool of query complexity and our aforementioned results. (Background on exact concept learning with queries appears in §2.3.1.) We prove results in two categories: (1) that classical queries in exact concept learning cannot be efficiently implemented in as queries to an unknown inductive invariant, and (2) that ICE-learning is provably harder than classical learning: namely, that, as advocated by Garg et al. [GLMN14], learning from counterexamples to induction is inherently harder than learning from examples labeled positive or negative.

3.6.1 Complexity Comparison

Table 3.1 collects our complexity results for invariant inference thus far, and compares them with complexity results for exact concept learning. The results of this section stem from this comparison.

In this chapter we have studied the complexity of inferring $\mathcal{L} = \text{Mon-CNF}_n$ invariants using Hoare/inductiveness queries in two settings: for *general systems* (in §3.4.1), and for *maximal systems* in §3.5. The results for exact concept learning displayed in Table 3.1 are for identifying the same class of formulas. For the sake of the comparison, the table maps inductiveness queries to equivalence queries (as these are similar at first sight) and maps the more powerful setting of Hoare queries to the more powerful setting of equivalence together with membership queries.

The comparison in the table of the complexity of inference with that of learning demonstrates that *invariant inference in general systems is harder* than exact learning; an inference algorithm needs to overcome not only the challenge of *space*, the complex syntactic structure of the invariant, but also the challenge of *time*, stemming from reachability in the transition system (see §1.2.1). The implications of the complexity gaps are elaborated in §3.6.2.

The complexity gap is eliminated when considering only *maximal systems*, alleviating the challenge of time. However, that is true only for the Hoare-query model, and gaps remain when considering only inductiveness queries; this is elaborated in §3.6.3.

Table 3.2: Concept vs. invariant learning: implementability of concept learning queries

	Maximal Systems		General Systems	
	Inductiveness	Hoare	Inductiveness	Hoare
Equivalence	✗	✓	✗	✗
Membership	✗	✓	✗	✗

3.6.2 Invariant Learning Cannot Be Reduced to Concept Learning

Table 3.2 summarizes our results for the possibility and impossibility of simulating concept learning algorithms in invariant learning. This table depicts implementability (✓) or unimplementability (✗) of membership and equivalence queries used in concept learning a class of formulas $\mathcal{L}$ through inductiveness and Hoare queries used in learning invariants for maximal systems over $\mathcal{L}$, and for general systems with candidate invariants in $\mathcal{L}$.

Formally, the implementability of query in a class of transition systems $\mathcal{P}$ and invariants $\mathcal{L}$ means that there is an algorithm that with a polynomial number of queries (in the respective model), given a transition system $TS \in \mathcal{P}$ that admits some (unknown) invariant $I \in \mathcal{L}$, correctly answers the query w.r.t. I:

- *Equivalence*: given a formula 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 $\sigma \not\models \theta, \sigma \models I$.
- *Membership*: given a state σ, return *true* if $\sigma \models I$ and *false* otherwise.

The proofs of impossibilities are based on the differences in complexity from Table 3.2. that neither equivalence nor membership queries can be simulated over general systems using even Hoare queries is implied by the hardness of general systems:

Corollary 3.6.1. *Equivalence queries cannot be efficiently implemented in the Hoare- or inductivenss-query model for general systems and* $\mathcal{L} = \text{Mon-CNF}_n$.

Proof. If it were, we could have simulate the algorithm for learning Mon-CNF_n which is subexponential [HKSS12] (or polynomial, with exponentially-long queries [Ang87b]), contradicting the exponential lower bound of Thm. 3.4.2. □

Corollary 3.6.2. *Membership queries cannot be efficiently implemented in the Hoare- or inductivenss-query model for general systems and* $\mathcal{L} = \text{Mon-CNF}_n$.

Proof. It it were, we could have efficiently implemented equivalence queries, which is impossible per Corollary 3.6.1: Perform an inductiveness check of θ. If there is a counterexample (σ, σ') perform a membership query on σ using Lemma 4.5.2: if the result is true, return σ' (a positive counterexample); otherwise return σ (a negative counterexample). □

When using only inductiveness queries, equivalence and membership queries cannot be implemented even over maximal systems:

Corollary 3.6.3. Equivalence *queries cannot be efficiently implemented in inductivenss-query model for* maximal *systems and* $\mathcal{L} = \text{Mon-CNF}_n$.

Proof. Similarly to the proof of Corollary 3.6.1, contradicting the exponential lower bound of Corollary 3.5.11. □

Corollary 3.6.4. Membership *queries cannot be efficiently implemented in the inductivenss-query model for* maximal *systems and* $\mathcal{L} = \text{Mon-CNF}_n$.

Proof. Follows from Corollary 3.6.3 similarly to how Corollary 3.6.2 follows from Corollary 3.6.1. □

The only possibility result in the table is of simulating inductiveness and membership queries using Hoare queries over maximal systems; the idea is that a Hoare query $\mathcal{H}(\delta_\varphi^{\mathcal{M}}, \mathit{Init}, \neg \mathit{cube}(\sigma)) \stackrel{?}{=} \mathit{false}$ implements a membership query on σ, thanks to fact that the inductive invariant is exactly the set of states reachable in one step. Such a membership query can be used to turn an inductiveness query into an implementation of an equivalence query, as explained in the proof of Corollary 3.6.2. Interestingly, the algorithm we use to show the polynomial upper bound on Hoare queries for maximal systems, PDR-1, is very similar to such a translation of an algorithm by Angluin [Ang87b] performing concept learning of Mon-CNF_n using equivalence and membership queries; we pick up on this idea in §4.5 of Chapter 4, with more sophisticated translations that are related to more realistic algorithms.

3.6.3 Counterexamples in Invariant Learning Are Inherently Ambiguous

As we have seen, equivalence queries cannot be implemented using inductiveness queries, even in the simple case of maximal systems (Corollary 3.6.3). The reason is that when the query fails—returns "not inductive" or "not equivalent"—then the counterexample provided to the inference algorithm is inherently weaker than the counterexample for the learning algorithm. In inference, the result is a counterexample to induction (an implication example, in the terminology of Garg et al. [GLMN14]), which is a *pair* of examples (σ, σ'), where σ is a negative example *or* σ' is a positive example, but there is no indication in the query itself of which is the case. In contrast, in classical equivalence queries, the counterexample is a single state σ, and it is in effect labelled—by checking whether the proposed candidate is satisfied by σ or not the learner can tell whether σ is a positive or negative example.

This discrepancy can be reformulated in the context of concept learning, as the difference between classical learning from equivalence queries (using labeled examples) and ICE learning [GLMN14], in which (essentially) the result of an equivalence query is an implication example. We have thus obtained a complexity result separating the two:

Corollary 3.6.5. *There exists a class of formulas* $\mathcal{L}$ *that can be learned using a subexponential number of equivalence queries, but requires an exponential number of ICE-equivalence queries.*

This result quantitatively corroborates the difference between *counterexamples to induction* and *examples labeled positive or negative*, a distinction advocated by Garg et al. [GLMN14].

3.7 Related Work for Chapter 3

Complexity of invariant inference. The fundamental question of the complexity of invariant inference in propositional logic has been studied by Lahiri and Qadeer [LQ09]. They show that deciding whether an invariant exists is PSPACE-complete. This includes systems with only exponentially-long invariants, which are inherently beyond reach for algorithms aiming to *construct* an invariant. In this thesis we focus on the search for polynomially-long invariants. Lahiri and Qadeer [LQ09] study the related problem of template-based inference, and show it is Σ_2^P-complete. Polynomial-length inference for CNF formulas can be encoded as specific instances of template-based inference; the Σ_2^P-hardness proof of Lahiri and Qadeer [LQ09] uses more general templates and therefore does not directly imply the same hardness for polynomial-length inference. The same work also shows that inference is only $\Pi_1^p = \mathbf{coNP}$-complete when candidates are only conjunctions (or, dually, disjunctions). In this thesis we focus on the richer classes of CNF or DNF invariants.

Black-box invariant inference. Black-box access to the program in its analysis is widespread in research on testing [e.g. ND12]. In invariant inference, Daikon [ECGN01] initiated the black-box learning of *likely* program invariants [see e.g. CTS08, SCIG08]. In this thesis we are interested in inferring necessarily correct inductive invariants. The ICE learning model, introduced by Garg et al. [GLMN14, GNMR16], and extended to general Constrained Horn Clauses in later work [END+18], pioneered a black-box view of inference algorithms such as Houdini [FL01] and symbolic abstraction [RSY04, TLLR15]. The inductiveness model in our work is inspired by this work, focusing on black-box access to the transition relation while providing the learner with full knowledge of the set of initial and bad states. Capturing PDR in a black-box model was achieved by extending ICE with relative-inductiveness queries [VGSM17]. Our work shows that an extension is necessary, and applies to any Hoare-query algorithm.

Lower bounds for black-box inference. To the best of our knowledge, our work provides the first unconditional exponential lower bound for rich black-box inference models such as the Hoare-query model. An impossibility result for ICE learning in polynomial time in the setting of quantified invariants was obtained by Garg et al. [GLMN14], based on the lower bound of Angluin [Ang90] for concept learning DFAs with equivalence queries. Our lower bound for monotone maximal systems (i) demonstrates an exponential gap between ICE learning and Hoare-query algorithms such as PDR (§3.5), and (ii) separates ICE learning from concept learning (§3.6); in particular, it holds even when candidates may be exponentially long (see Corollary 3.5.11 and [GLMN13, Appendix B]).

Learning and synthesis with queries. The lens of synthesis has inspired many works applying ideas from machine learning to invariant inference [e.g. JGST10, SGH+13b, SNA12,

SGH+13a, GLMN14, SA16]. The role of learning with queries is recognized in prominent synthesis approaches such as Counterexample-Guided Inductive Synthesis (CEGIS) [STB+06] and synthesizer-driven approaches [e.g. Gul12, JGST10, LPP+17], which learn from equivalence and membership queries [JS17, ABD+15, BDVY17, DSY17]. The theory of oracle-guided inductive synthesis [JS17] theoretically studies the convergence of CEGIS in infinite concept classes using different types of counterexamples-oracles, and relates the finite case to the teaching dimension [GK95]. In this work we study inference based on a different form of queries, and prove lower bounds on the convergence rate in finite classes. Additional notable applications in formal methods of exact concept learning include the use of the L^* algorithm [Ang87a] for synthesizing assumptions and guarantees in compositional reasoning [AMN05, ACMN05].

Proof complexity. Proof complexity studies the power of polynomially-long proofs in different proof systems. A seminal result is that a propositional encoding of the pigeonhole principle has no polynomial resolution proofs [Hak85]. Ideas and tools from proof complexity have been applied to study SAT solvers [e.g. PD11] and recently also SMT [RKG18]. Proof complexity is an alternative technical approach to study the complexity of proof search algorithms, by showing that some instances do not have a short proof, showing a lower bound regardless of how search is conducted. Our work, inspired by learning theory, provides exponential lower bounds on query-based search even when the proof system is sufficiently strong to admit short proofs: in our setting, there is always a short derivation of an inductive invariant by generalization in backward reachability, blocking counterexamples with the optimal choice, using clauses from a target invariant (see §3.4.3). We expect that proof complexity methods would prove valuable in further study of inference.

Chapter 4

Upper Bounds for Interpolation-Based Invariant Inference

This chapter is based on the results published in [FSSW21, FS22].

In this chapter we study the complexity of learning invariants in the interpolation-based approach and its connections to exact concept learning. We define a condition on invariants and their geometry, called the *fence* condition, by which it is possible to apply 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 results in concept learning to efficiently infer invariants of larger syntactic classes of invariants beyond monotone DNF, through general transformations from exact concept learning algorithms, as well as specially-crafted invariant inference algorithms that employ Bshouty's monotone theory, including a novel algorithm for efficiently computing monotonization. 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.

The upper bounds of this chapter go beyond the class of antimonotone maximal system for which an upper bound (Corollary 3.5.8) was shown in Chapter 3, in that in the results of this chapter, the diameter of the system need not be one, not all states in the invariant need to be k-reachable, and the invariant can be richer than antimonotone CNF or monotone DNF.

4.1 Overview

```
init  x_1 = ... = x_n = true

repeat:
      | hot_potato()
      | turn_two_on()
      | havoc_others()

   assert ¬(x_1 = false, x_2 = ... = x_n = true)

havoc_others():
   forall x_i ∉ J:
      x_i := *
```

```
hot_potato():
   let x_i ≠ x_j ∈ J
   if x_i = false ∧ x_j = true:
      x_i := true
      x_j := false

turn_two_on():
   let x_i ≠ x_j ∈ J
   if x_i = false ∧ x_j = false:
      x_i := true
      x_j := true
```

Figure 4.1: 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. $\texttt{x} := *$ means that `x` is updated to either true or false nondeterministically.

Suppose we would like to automatically prove the safety of the transition system in Fig. 4.1. 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 $x_1 = \mathit{false}$ 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 the safety of the transition system. One inductive invariant here is that the variables in J are always *true*:

$$I = \bigwedge_{x_i \in J} x_i. \tag{4.1}$$

(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 [McM03]. 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 [FQ02, GS97].

4.1.1 Interpolation-Based Invariant Inference

We start our investigation with the pioneering interpolation-based algorithm [McM03], depicted in Alg. 2. The algorithm operates by forward exploration, in each iteration weakening (adding more states) to the current candidate φ, starting from $\varphi = \mathit{Init}$. The algorithm chooses an

unrolling bound k, and asserts that φ cannot reach *Bad* in k steps (line 4). If the check fails, φ 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 $\varphi = \mathit{Init}$, 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 φ, namely, includes all states that are reachable in one step from φ; 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 [Cra57] of a bounded model checking (BMC) query, which can be computed from the SAT solver's proof [McM03]. 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 2 Interpolation-based invariant inference [McM03]

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

Algorithm 12 Interpolation by term minimization [CIM12, BGKL13]

```
1: procedure TERMMINITP(φ, δ, Bad, k − 1)
2:     χ ← false
3:     while δ(φ) ⟹̸ χ do
4:         let σ′ such that σ′ ∈ δ(φ), σ′ ⊭ χ
5:         if δ̲^{k−1}(σ′) ∩ Bad ≠ ∅ then fail
6:         d ← cube(σ′)
7:         for ℓ in d do
8:             if δ̲^{k−1}(d \ {ℓ}) ∩ Bad = ∅ then
9:                 d ← d \ {ℓ}
10:        χ ← χ ∨ d
11:    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 [McM03]. 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" [McM18]. 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. 2 by McMillan [McM03], 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 φ 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 φ is always strictly increasing (becoming strictly weaker) and there is only a finite (albeit exponential) number of strictly increasing formulas, Alg. 2 must converge, and at that point φ 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. 4.1, the states that can reach *Bad* are the states where there is an odd number of *false* variables in J. The gfp is thus the invariant saying that the number of *false* variables in J is even, whose minimal DNF representation is exponentially long [CH11, 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. 4.1, 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 [McM03]. 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*

[1] McMillan [McM03] calls this number the "reverse-depth".

[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_on` 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*.

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. 2 itself is not amenable to such an analysis, because valid choices for χ range between the exact post-image $\delta(\varphi)$ 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. 12, due to Chockler et al. and Bjørner et al. [CIM12, BGKL13] and inspired by IC3/PDR [Bra11, EMB11]. The procedure iteratively samples states from the post-image of φ 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 $\delta(\varphi)$ 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.

4.1.2 The Complexity of Interpolation-Based Inference

The interpolation-based inference of Alg. 2 with the interpolation procedure of Alg. 12 produce invariants in disjunctive normal form (DNF). (For example, the invariant in Equation (4.1) is a DNF formula with one term.) If the shortest invariant in DNF is exponentially long, clearly an exponential 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.

[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. [CIM12] found that the extra time spent in explicit minimization after the unsat core is compensated by fewer iterations of the overall algorithm.

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 (4.1) consists of the states where $|\{i \in J \mid x_i = \mathit{false}\}| = 1$: 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. 1.1 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. 4.1, the invariant in Equation (4.1) 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 §4.2. (See Fig. 1.2 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 4.2.4). 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]

The fence condition is a property of transition systems that does not hold in general, and thus allows to evade the lower bounds of Chapter 3 (§3.4).

Convergence With the Fence Condition and a Sufficient BMC Bound

Intuitively, when the fence condition does not hold, if Alg. 12 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

[4]Even for the co-diameter, the existing analysis by McMillan [McM03] does not derive complexity bounds, while we are able to do so using techniques from exact concept learning.

σ^- does not reach *Bad* in k steps we could include σ^- in the candidate even though it is not in I. In §4.3.2, 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 returns, and *regardless* of the order in which Alg. 12 attempts to drop literals (Lemma 4.3.1)—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. 4.1, 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.

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 §4.3.3 that if I is in monotone DNF, that is, all variables appear positively, then the number of iterations of Alg. 2 is bounded by the number of terms in I if I is k-fenced (Thm. 4.3.5)—thus addressing question (2) above. ***Overall, this is a theorem of efficient invariant inference by Alg. 2.*** For example, the system in Fig. 4.1 admits the short monotone DNF invariant in Equation (4.1), 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 antimonotone CNF (Corollary 4.3.9)—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*.

4.1.3 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 §4.4, 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 4.4.3). This upper bound is achieved by a different, yet related algorithm, which is based on the celebrated work of Bshouty [Bsh95] in machine learning. Roughly, the algorithm uses several instances of Alg. 2 that learn several overapproximations of I, each of them monotone under a different translation of the variables. A dual result holds for almost-antimonotone CNF invariants and the dual fence condition (Corollary 4.4.4).

Underlying this development is the realization that efficient inference based on the fence condition has close connections to efficient *exact concept learning* [Ang87b]. 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 [GLMN14, FISS20]. Nevertheless, we show that the algorithm which achieves Corollary 4.4.3 can be obtained directly from the exact concept learning algorithm by Bshouty [Bsh95] through a transformation which, *when the fence condition holds*, can implement *certain* equivalence and membership queries using BMC (§4.5.1). 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 [Bsh95], in §4.4.1.) It is interesting to note that applying this transformation to an existing concept learning algorithm [Val84, Ang87b, AP95] produces an algorithm that matches Alg. 2+Alg. 12.

Our transformation from exact concept learning to invariant inference places restriction on the learning algorithm. A particularly interesting class of invariants are those 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, but Bshouty's algorithm for efficiently learning such formulas [Bsh95] does not satisfy the premise of our transformation. Another translation we provide does apply more broadly, to *every* learning algorithm, but the resulting invariant inference algorithm requires a stronger condition, a *two-sided* fence condition (§4.5.2). To this end, we develop a novel invariant inference algorithm that efficiently infers invariants with short CNF and DNF representations based on the standard, *one-sided* fence condition, in a way that is inspired by the learning algorithm but deviates from it in significant ways (§4.6).

4.1.4 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 can 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 be a successful strategy to achieve convergence [SRW02, FWSS19, SDDA11, BBL+17, Nam07, CBKR19]. The effect of program transformations on inference depends on the inference algorithm.[5] In §4.7 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. 4.1 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 $x_1 = \mathit{false}, x_2 = \ldots = x_n = \mathit{true}$, *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 (4.1) 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 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.

[5] For example, the inference of conjunctive invariants via Houdini [FL01] is robust, and always finds an invariant if one exists in a polynomial number of SAT calls.

Notation. In this chapter 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.

4.2 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 4.2.1 (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 4.2.2 (Backwards k-Fenced)**.** *For a transition system $(Init, \delta, Bad)$, an inductive invariant I is* backwards k-fenced *for $k \in \mathbb{N}$ if $\partial^-(I) \subseteq \left(\underline{\delta^{-1}}\right)^k (Bad)$.*

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

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 $even(\mathtt{x})$ holds is bad, and every state where $odd(\mathtt{y})$ holds is either bad, or the step that adds $\mathtt{y}$ to $\mathtt{x}$ leads to a bad state. Hence, I is 1-backwards fenced.

Now consider the same system except there is a flag $\mathtt{z}$ that decides whether $\mathtt{x}$ is modified; the system takes a step only if $\mathtt{z} = false$. The same invariant from before applies, but it is no longer backwards 1-fenced. This is because the state in $\partial^-(I)$ where $\mathtt{x}$ is odd, $\mathtt{y}$ is odd, and $\mathtt{z} = false$ cannot reach a bad state (nor perform any transition). However, a different invariant, $odd(\mathtt{x}) \wedge (even(\mathtt{y}) \vee \neg\mathtt{z})$ 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 4.2.4. *Every safe transition system $TS = (Init, \delta, Bad)$ admits an inductive invariant $gfp = \neg\left(\left(\underline{\delta^{-1}}\right)^{\omega}(Bad)\right)$ that is backwards k-fenced for k that is the co-diameter: the minimal k such that $\left(\underline{\delta^{-1}}\right)^{k}(Bad) = \left(\underline{\delta^{-1}}\right)^{\omega}(Bad)$.*

Proof. First, note that such a k exists because for every $\sigma \in \left(\underline{\delta^{-1}}\right)^{\omega}(Bad)$ there is a finite r such that $\sigma \in \left(\underline{\delta^{-1}}\right)^{r}(Bad)$, and k is the maximum over these r's, and there are finitely many of these because the number of states is finite.) By the choice of I it holds that $\neg I \subseteq \left(\underline{\delta^{-1}}\right)^{\omega}(Bad)$. Seeing that $\partial^{-}(I) \subseteq \neg I$, we have obtained $\partial^{-}(I) \subseteq \left(\underline{\delta^{-1}}\right)^{k}(Bad)$, as desired. □

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 §5.1. An additional example follows.

Example 4.2.5. *Consider an example of a (doubly)-linked list traversal, using* `i` *to traverse the list backwards, modeled via predicate abstraction following Itzhaky et al. [IBR+14]. 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* $\mathtt{s} \rightsquigarrow r$ *to denote that r is reachable from* `s` *by following zero or more links. Consider the initial assumption* $\mathtt{h} \rightsquigarrow \mathtt{x}$*, but* $\mathtt{i} \not\rightsquigarrow \mathtt{x}$ *(it may be that* $\mathtt{x} \rightsquigarrow \mathtt{i}$*, or that* `i` *is not at all in the list). The bad states are those where* $\mathtt{i} = \mathtt{h}$*.*

An inductive invariant for this system is $\mathtt{h} \rightsquigarrow \mathtt{x} \wedge \neg(\mathtt{i} \rightsquigarrow \mathtt{x})$*. In predicate abstraction, we may take the predicates $p_{h,x} = \mathtt{h} \rightsquigarrow \mathtt{x}$, $p_{i,x} = \mathtt{i} \rightsquigarrow \mathtt{x}$, 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) $p_{h,x} = false, p_{i,x} = false$ and (2) $p_{h,x} = true, p_{i,x} = true$. Both states are in fact bad states under the abstraction: both include a state where* $\mathtt{i} = \mathtt{h}$*, 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 $p_{h,x} = false, p_{i,x} = true$ abstracts only states where $\mathtt{h} \not\rightsquigarrow \mathtt{i}$*, 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).*

4.3 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. §4.3.1 describes the algorithm. §4.3.2 derives the algorithm's basic properties and its convergence from the fence condition. §4.3.3 builds on this to obtain the efficiency result.

4.3.1 Interpolation by Term Minimization

We begin with a formal presentation of the algorithm we will be analyzing in this section, Alg. 13. It is a simplification of the one presented in §5.1, "merging" the two loops formed when Alg. 2 uses Alg. 12; instead of first computing an overapproximation χ of the post-image of the entire previous candidate and only then disjoining χ, Alg. 13 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. 2+Alg. 12.

Algorithm 13 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 δ̲^k(σ') ∩ Bad ≠ ∅ then
6:              restart with larger k
            d ← cube(σ')
7:          for ℓ in d do
8:              if δ̲^k(d \ {ℓ}) ∩ Bad = ∅ then
9:                  d ← d \ {ℓ}
10:         φ ← φ ∨ d
11:     return I
```

Alg. 13 starts with the candidate invariant $\varphi = \mathit{Init}$, 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 $\mathit{cube}(\sigma')$—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.

4.3.2 Interpolation Confined in the Boundary

We now show how the fence condition ensures that Alg. 13 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 §4.3.3 to obtain the efficiency result.

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

Lemma 4.3.1. *Let $(\mathit{Init}, \delta, \mathit{Bad})$ 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*$(\mathit{Init}, \delta, \mathit{Bad}, k)$, every candidate φ is an underapproximation of I, namely, $\varphi \Longrightarrow I$.*

Proof. By induction on the algorithm's iterations: initially, $\varphi = \mathit{Init} \Longrightarrow I$ since I is an inductive invariant; for later iterations, we show that every $d \Longrightarrow I$. By induction on the iterations in generalization: when it starts, $d = \sigma'$ satisfies this, because $\varphi \Longrightarrow I$ so $\sigma \models I$, hence also

$\sigma' \models I$ because I is inductive. Later, assume there is a point when $d \Longrightarrow I$ stops holding: the algorithm drops a literal ℓ, obtaining $\tilde{d} = d \setminus \{\ell\}$ where $d \Longrightarrow I$ but $\tilde{d} \not\Longrightarrow I$, and the check passes: $\underline{\delta}^k(\tilde{d}) \cap Bad = \emptyset$. 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 $d \Longrightarrow I$), 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 $\underline{\delta}^k(\tilde{d}) \cap Bad = \emptyset$ 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 §4.1.1), 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 §4.3.3.)

Lemma 4.3.2. *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 $\varphi \Longrightarrow I$.*

Proof. We first claim that no execution fails (line 6 of Alg. 13 does not execute). As in the proof of Lemma 4.3.1, $\sigma' \models I$ as $\varphi \Longrightarrow I$ and I is inductive. Since I is an inductive invariant, σ' cannot reach Bad in any number of steps, and in particular $\sigma' \notin \left(\underline{\delta^{-1}}\right)^k (Bad)$.

Since φ grows monotonically, and there are only finitely many formulas over the fixed vocabulary, φ must converge. When this happens I is inductive invariant, and we have $\varphi \Longrightarrow I$ by Lemma 4.3.1. □

Recalling that the gfp is backwards k-fenced for k which is at most the co-diameter (Lemma 4.2.4), this yields a completeness result, akin to the completeness result by McMillan [McM03].

Corollary 4.3.3. *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 4.3.4. *What happens to Alg. 13 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 4.3.1), 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 4.3.1) even when assuming that* any *$\sigma^+ \models I$ can be discovered this way.*

4.3.3 Inference of Monotone Invariants

In this section we prove that Alg. 13 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 4.3.1), but exponentially many iterations could be required before the algorithm converges; we show that this cannot happen with monotone DNF invariants.

We now leverage this for the analysis of Alg. 13. 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 4.3.5. *Let $(Init, \delta, Bad)$ be a transition system and $k \in \mathbb{N}$. If there is an inductive invariant $I \in$ 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.*

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 4.3.1. 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, $\underline{\delta}^k(d \setminus \{\ell\}) \cap Bad = \emptyset$, and in particular $\underline{\delta}^k(\tilde{d} \setminus \{\ell\}) \cap Bad = \emptyset$. 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 4.3.2 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 I, because it is a prime implicant of I and I is monotone and by Thm. 2.3.3. 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 4.3.1, $I \equiv \varphi$ 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 4.3.6. *Thm. 4.3.5 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. 4.3.5, 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 4.3.7. *What happens when Alg. 13 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 antimonotone CNF invariants. An efficiency result for inferring antimonotone *CNF* invariants that satisfy the *forward* fence condition follows, through the duality discussed in §2.2.1. The definition of antimonotone CNF formulas appears in Def. 2.3.7.

Definition 4.3.8 (Forwards k-Fenced)**.** *For a transition system $(Init, \delta, Bad)$, an inductive invariant I is* forwards k-fenced *for $k \in \mathbb{N}$ if $\partial^+(I) \subseteq \underline{\delta}^k(Init)$.*

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. 4.3.5 we obtain:

Corollary 4.3.9. *Let $(Init, \delta, Bad)$ be a transition system and $k \in \mathbb{N}$. If there exists an inductive invariant $I \in$ Mon-CNF$_m$ that is forwards k-fenced, then* Dual-MB-ITP = ITP-Inference-TermMin**$(Init, \delta, Bad)$, 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 §4.6.2.

Remark 4.3.10. *Thm. 4.3.5 and Corollary 4.3.9 can be extended to the case of unate DNF (Def. 2.3.4) when the backwards/forwards fence condition holds, using essentially the same proof as in Thms. 2.3.3 and 4.3.5, replacing Thm. 2.3.3 by Corollary 2.3.5.*

Alg. 13 beyond monotone invariants. What happens when we try to apply Alg. 13 to infer invariants that satisfy the fence condition but are not monotone (or unate)? By Lemma 4.3.1, the algorithm would converge to an inductive invariant—but this make take an exponential number

of iterations. In Thm. 4.3.5 we have shown that each iteration produces a prime implicant of I. Unfortunately, the number of prime implicants can be exponential [CM78, SST08]. Worse, Aizenstein and Pitt [AP95] 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).

4.4 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 [Bsh95] (see §2.4.4). We first present the invariant inference algorithm and a self-contained proof of efficient convergence relying on the fence condition. In §4.5.1 we obtain an alternative proof by a transformation that can "simulate" the original algorithm through the fence condition. The goal of the explicit exposition in this section is twofold: first, highlighting "family resemblance" between this algorithm and the model-based interpolation-based algorithm from the previous sections; second, serving as a first step towards the algorithm for CDNF invariants in §4.6, that expands on the same ideas in new ways.

Algorithm 14 Interpolation-based inference using a known monotone basis (building on [Bsh95])

```
1:  Assuming a known basis {b_1, ..., b_t} (Def. 2.4.17)
2:  procedure ITP-KnownMBasis(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', b_i, δ, Bad, k)
10:                 H_i ← H_i ∨ d
11:     return ⋀_{i=1}^{t} H_i
12: procedure MonGenBmc(σ, b, δ, Bad, k)
13:     if δ̲^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 b[p_j] ≠ v[p_j] do
19:             x ← v[p_j ↦ b[p_j]]
20:             if δ̲^k(x) ∩ Bad = ∅ then
21:                 v ← x; walked ← true
22:     return cube_b(v)
```

4.4.1 Inference with a Monotone Basis

In this section we present Alg. 14, 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 §4.4.2. We use, and recall along the way, notions from the monotone theory [Bsh95]; see §2.4 for a detailed exposition.

Recall that a formula is b-monotone (Def. 2.4.2), where b is a state, if it can be written in DNF so that every variable p appears in polarity $\neg b[p]$ (Lemma 2.4.6 and Corollary 2.4.11).

In general, a formula may not be b-monotone for any b; but it can always be expressed as a conjunction of formulas, each monotone w.r.t. some valuation. A set of states that suffices for this is called a basis (Def. 2.4.17).

Alg. 14 infers an invariant under the assumption that a known set of states is a basis for the invariant. (We address the choice of the basis in §4.4.2.) The main idea of Alg. 14 is to think about the desired invariant I as a conjunction $\bigwedge_{i=1}^{t} H_i$ where each H_i is b_i-monotone, and infer the H_i formulas. The first concern is to whether such H_i always exist in a way that can be represented compactly; the first insight is that $\mathcal{M}_{b_i}(I)$, the least b_i-monotone overapproximation of I (Def. 2.4.3), is a good choice: by Lemma 2.4.7, $\mathcal{M}_{b_i}(I)$ has a DNF representation that is not larger than the shortest DNF representation of I.

Our goal now is to gradually infer $\mathcal{M}_{b_i}(I)$, 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 σ' must be added to every H_i that does not yet include it. To do so, we add (disjoin) a b_i-monotone term to H_i. The term we add (ignoring generalization at this point) is the monotone cube (see Corollary 2.4.8), the least b_i-overapproximation of σ' (as a full cube), which explicitly is

$$\mathit{cube}_{b_i}(\sigma') = \bigwedge \{p \mid \sigma'[p] = \mathit{true},\ \sigma'[p] \neq b_i[p]\} \wedge \bigwedge \{\neg p \mid \sigma'[p] = \mathit{false},\ \sigma'[p] \neq b_i[p]\}.$$

The monotone cube includes more states than the original σ', but by Corollary 2.4.8 this cannot overgeneralize beyond $\mathcal{M}_{b_i}(\varphi)$, and $\mathit{cube}_{b_i}(\sigma') \Longrightarrow \mathcal{M}_{b_i}(I)$ provided that $\sigma' \models I$.

Adding the monotone cube is thus "safe", but may converge to $\mathcal{M}_{b_i}(I)$ too slowly. Lemma 2.4.7 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 $\mathcal{M}_{b_i}(I)$ 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 b_i, the smaller the conjunction in the $\mathit{cube}_{b_i}(v)$ is. In fact, this minimization successfully achieves an actual term of $\mathcal{M}_{b_i}(I)$. This is established in the next lemma, akin to Thm. 2.3.3 in the purely monotone case.

Lemma 4.4.1 ([Bsh95], Proposition A + Lemma 1(1))**.** *A state x is b-minimal positive for φ if $x \models \varphi$ and for every i such that $x[p] \neq b[p]$ it holds $x[p \mapsto b[p]] \not\models \varphi$. Let x be b-minimal positive for a formula φ in DNF. Then there is a term t of φ such that $\mathcal{M}_b(t) \equiv \mathit{cube}_b(x)$.*

Proof. Since $x \models \varphi$ which is in DNF, there is a term t of φ such that $x \models t$. From Corollary 2.4.8, $\mathit{cube}_b(x) \Longrightarrow \mathcal{M}_b(t)$. For the other direction, we need to show that the $\mathcal{M}_b(t)$, which is a conjunction by Lemma 2.4.7, includes all the conjuncts in $\mathit{cube}_b(x)$. To this end, let p be such that $x[p] \neq b[p]$; we need to show that p is a literal of t if $x[p] = \mathit{true}$, and $\neg p$ is a literal of t if $x[p] = \mathit{false}$. Suppose otherwise. Then $x[p \mapsto \neg x[p]]$ also satisfies t. But then $x[p \mapsto b[p]] = x[p \mapsto \neg x[p]] \models \varphi$, in contradiction to the premise. □

The algorithm: ITP-KnownMBasis. We now collect the ideas from above and describe the algorithm (Alg. 14) for inferring invariants that admit a known monotone basis, based on the backwards fence condition. Assuming a known basis $\{b_1, \ldots, b_t\}$ for the target (unknown) invariant I, the algorithm maintains a sequence $H_1, \ldots, H_t$, where each H_i is an b_i-monotone DNF formula. Each H_i is gradually increased until it is $\mathcal{M}_{b_i}(I)$ (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. 13), each for learning an b_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 $\mathcal{M}_{b_i}(I)$, the algorithm gradually flips bits in the state that disagree with b_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 4.4.1; 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 $\{b_1, \ldots, b_t\}$ is not a basis for I), because it checks that H is inductive before returning; if H is 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 $\{b_1, \ldots, b_t\}$ is a basis:

Theorem 4.4.2. *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, $I \in \mathrm{DNF}_m$, and $\{b_1, \ldots, b_t\}$ is a monotone basis for I (Def. 2.4.17), then* ITP-KnownMBasis*$(Init, \delta, Bad, k)$ 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 $H_i \implies \mathcal{M}_{b_i}(I)$ and H_i is b_i-monotone (for every i). From this it would follow that $\bigwedge_{i=1}^{t} H_i \implies I$, because $I = \bigwedge_{i=1}^{t} \mathcal{M}_{b_i}(I)$ from the basis assumption and Thm. 2.4.19. From this it would follow that the counterexample is always positive, $\sigma' \models I$. This implies that $\underline{\delta}^k(\sigma') \cap Bad = \emptyset$, 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. $\underline{\delta}^k(x') \cap Bad = \emptyset$. 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 Corollary 2.4.8, $cube_{b_i}(x) \implies \mathcal{M}_{b_i}(I)$ 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 $x[p] \neq b_i[p]$ we have $\underline{\delta}^k(x[p \mapsto b_i[p]]) \cap Bad \neq \emptyset$, and in particular $x[p \mapsto b_i[p]] \not\models I$. Using Lemma 4.4.1, $cube_{b_i}(x)$ is a term of the DNF representation of $\mathcal{M}_{b_i}(I)$ from Lemma 2.4.7. By Lemma 2.4.7 this representation has m terms. Overall we need at most $t \cdot m$ iterations. □

Between Algs. 13 and 14. Alg. 14 has the same backbone as the model-based interpolation-based algorithm we studied earlier. Both algorithms iteratively obtain counterexamples to induction and use the post-states as positive examples, to be included in the next candidate invariant. Both algorithm generalize the counterexample and include not just the positive example itself but also many other related states, relying on a BMC heuristic, of adding as many states as possible—within a predefined space of viable possibilities—without adding states that reach *Bad* in k steps. In Alg. 13, the space of possible generalizations is the set of terms that are derived from the example state. Alg. 14 manipulates terms, but more than one—the algorithm computes a generalization term for each element in the monotone basis. In both algorithms, generalizing a term amounts to dropping literals while the BMC criterion still holds. The query in Alg. 14 is slightly different, performing BMC on a single state rather than a set of states (and similar to how this is performed in Angluin's algorithm for exact learning monotone formulas, see §2.3.2); the reason is that each b_i-monotone hypothesis H_i might actually include states that can reach *Bad* in k steps (they will be excluded by other H_i's).

4.4.2 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 states with at most r variables assigned *true* is a basis for this class [Bsh95]. 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 b-monotone for the state b 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 [Bsh95].

Applying Thm. 4.4.2 with the known basis for r-almost-monotone DNF yields:

Corollary 4.4.3. *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* ITP-KnownMBasis*$(Init, \delta, Bad, k)$ 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 antimonotone CNF invariants, which are CNF formulas with at most r clauses that include positive literals, is as follows:

Corollary 4.4.4. *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-antimonotone CNF with m clauses, then* ITP-KnownMBasis* *$(Init, \delta, Bad, k)$ 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.*

4.5 From Exact Learning to Invariant Inference via the Fence Condition

Exact learning with queries [Ang87b] 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 §4.5.1 we obtain Algs. 13 and 14 by a translation from exact learning algorithms that satisfy certain restrictions. In particular, this provides an alternative proof of Thm. 4.4.2. In §4.5.2 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. 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, as we showed in Chapter 3 (§3.6), and here rely on the fence condition. (Background on exact concept learning with queries appears in §2.3.1.)

4.5.1 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 [GLMN14], and, as we have shown in §3.6.2, is unable to efficiently implement a classical teacher. 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 (Corollary 3.6.2). 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 (Corollary 3.6.1): 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 4.5.1 (Implementing positive equivalence queries)**.** *Let $(Init, \delta, Bad)$ be a transition system and I an inductive invariant. Given θ such that $\theta \Longrightarrow I$, 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 4.5.2 (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 $\underline{\delta}^k(\sigma) \cap Bad = \emptyset$.*

This is a special case of an implementation of certain subset queries (a membership query is with $\theta = \{\sigma\}$):

Lemma 4.5.3 (Implementing positive-adjacent subset queries)**.** *Let $(Init, \delta, Bad)$ be a transition system and I an inductive invariant that is backwards k-fenced. Given θ s.t. $\theta \subseteq I$ or $\theta \cap \partial^-(I) \neq \emptyset$, it is possible to decide whether $\theta \subseteq I$ using a single k-BMC check of whether $\underline{\delta}^k(\theta) \cap Bad = \emptyset$.*

Proof. If $\theta \subseteq I$, no state in θ can reach Bad in any number of steps, k in particular, and we correctly return *true*. Otherwise, from the premise, there is $\sigma \in \theta$ s.t. $\sigma \in \partial^-(I)$ so, by the fence condition, we must have $\underline{\delta}^k(\sigma) \cap Bad \neq \emptyset$, hence also $\underline{\delta}^k(\theta) \cap Bad \neq \emptyset$, and we correctly return *false*. □

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

Corollary 4.5.4. *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 subset queries (including membership queries). Assume further that when $\mathcal{A}$ performs an equivalence query on θ, always $\theta \Longrightarrow \varphi$, and when $\mathcal{A}$ performs a subset query on θ, always $\theta \subseteq \varphi$ or $\theta \cap \partial^-(\varphi) \neq \emptyset$. 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, s_2 k-BMC checks, and time the same as of $\mathcal{A}$ up to a constant factor.*

Proof. We simulate $\mathcal{A}$ using the equivalence queries from Lemma 4.5.1 and the subset queries from Lemma 4.5.3. 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". The time overhead of performing each inductiveness and subset queries according to Lemmas 4.5.1 and 4.5.3 is a constant factor. □

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

Alg. 13 and exact learning. The interpolation algorithm of Alg. 13 and its efficiency result (Thm. 4.3.5) can be obtained by the transformation of Corollary 4.5.4 from the exact learning algorithm for DNF formulas as it appears in Alg. 3 in §2.3.2, and its efficiency result for monotone DNF formulas (Thm. 2.3.6) . The queries performed in Alg. 3 satisfy the conditions of the transformation: as the algorithm's analysis shows (see Thm. 2.3.6), the hypothesis is always below the true formula, as required for equivalence queries; the subset queries are always positive adjacent, because if d is a term s.t. $d \subset \psi$, and $d' \not\subseteq \psi$ where $d' = d \setminus \{\ell\}$, then taking a state $\sigma^- \in d' \setminus \psi$ and flipping the variable in ℓ results in a state $\sigma^+ \models d$ and hence $\sigma^+ \models \psi$, hence $\sigma^- \in \partial^-(\psi)$ and $\sigma^- \in d'$, as required.

Alg. 14 and exact learning. The translation in Corollary 4.5.4 provides an alternative proof of Thm. 4.4.2.

Proof of Thm. 4.4.2. Alg. 14 is obtained by the transformation in Corollary 4.5.4 applied on the Λ-algorithm for exact concept learning using a known monotone basis by Bshouty [Bsh95, §5], whose code is presented in Alg. 5 in §2.4.4. 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 4.5.4. Indeed, the hypothesis is always below the true formula and counterexamples are always positive [Bsh95, §5.1.1, inductive property 1], and membership queries are always performed after flipping one bit in a positive example, yielding positive-adjacent membership queries per the requirement of Lemma 4.5.2. □

4.5.2 Inference From Two-Sided Fence and Exact Learning

In this section we simulate arbitrary exact learning algorithms (going beyond the requirements in Corollary 4.5.4) relying on a *two-sided* fence condition. An important example of such an exact learning algorithm is the CDNF algorithm by Bshouty [Bsh95]. The conditions of the transformation in §4.5.1 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 4.5.5. *In the example of Fig. 4.1, the invariant in Equation (4.1) is backwards* 2*-fenced (see §4.1.2). 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 4.5.6 (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.*[6]

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

At least one of the queries is true at some point, because $\sigma_0 \in \underline{\delta}^{k_1}(Init)$.

Suppose $\sigma \in I$. Then in this process, as long as σ stays in I, we cannot return *false* because states in I do not reach Bad (in k_2 steps or more). Then either σ always stays in I, in which case we return *true* when σ becomes σ_0, or there is a first crossing point from $\sigma_1 \in I$ to $\sigma_2 \notin I$, where, from the premise that I is forwards k_1-fenced, $\sigma \in \underline{\delta}^{k_1}(Init)$ and we return *true*, as expected.

Suppose $\sigma \notin I$. Then in this process, as long as σ stays *not* in I, we cannot return *true*, because states in $\neg I$ are not reachable from $Init$ (in k_1 steps or more). Because we end the process with $\sigma_0 \in I$ there must be a first crossing point from $\neg I$ to I, where, from the premise that I is backwards k_2-fenced, $\underline{\delta}^{k_2}(\sigma) \cap Bad \neq \emptyset$, and we return *false*, as expected. □

An equivalence query can be implemented by an inductiveness check and a membership query (as was also noted in §3.6.2, in Corollary 3.6.2):

Lemma 4.5.7 (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.*

Proof. Perform an inductiveness check of θ. If there is a counterexample (σ, σ') perform a membership query on σ using Lemma 4.5.2: if the result is true, return σ' (a positive counterexample); otherwise return σ (a negative counterexample). □

[6] 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.

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

Corollary 4.5.8. *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, $n(s_1 + s_2)$ of k_2-BMC checks, and time $O(n(s_1 + s_2)t_{\mathcal{A}})$ where $t_{\mathcal{A}}$ is the worst-case time of $\mathcal{A}$ concept-learning I.*

Proof. We simulate $\mathcal{A}$ using the equivalence queries from Lemma 4.5.7 and the membership queries from Lemma 4.5.6. 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. For soundness, we always verify the result before returning using an additional inductiveness check. The time overhead corresponds to the multiple BMC queries required instead of a single membership or equivalence query. □

Next, we demonstrate an application of Corollary 4.5.8 to the inference of a larger class of invariants.

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 4.4.3 and 4.4.4). We now utilize Corollary 4.5.8 to the CDNF algorithm by Bshouty [Bsh95] 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 4.5.9. *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 $I \in \mathrm{DNF}_{m_1}$, $I \in \mathrm{CNF}_{m_2}$, 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.*

As noted by Bshouty [Bsh95], the class of formulas with short DNF and CNF includes the formulas that can be expressed by a small *decision tree*: a binary tree in which every internal node is labeled by a variable and a leaf by *true*/*false*, and σ satisfies the formula if the path defined by starting from the root, turning left when the σ assigns *false* to the variable labeling the node and right otherwise, reaches a leaf *true*. The size of a decision tree is the number of leaves in the tree.

Corollary 4.5.10. *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 that can be expressed as a decision tree of size m, and I is both backwards- and forwards- k-fenced, then $\mathcal{A}(Init, \delta, Bad, k)$ converges to an inductive invariant in $O(m^2)$ inductiveness checks, $O(m^2 \cdot n^3)$ k-BMC checks, and $O(m^2 \cdot n^3)$ time.*

Proof. A decision tree of size m has a DNF representation of m terms: a disjunction of terms representing the paths that reach a leaf *true*, each is the conjunction of the variables on the path with polarity according to left/right branch. Similarly, it has a CNF representation of m clauses: a conjunction of clauses which are the negations of paths that reach a leaf *false*. Now apply Thm. 4.5.9. □

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 [Bsh97]. Whether this is possible based on the one-sided fence condition is an interesting question for future work.

In the next section we will show that this can be done for CDNF invariants (and, in particular, decision trees)—infer them based on only the one-sided version of the fence condition. In the CDNF case, the transformation of Corollary 4.5.4 that uses only the one-sided fence condition is inapplicable to the Bshouty's algorithm for learning CDNF (Alg. 6) because it performs equivalence queries that are not one-sided,[7] necessitating a new inference algorithm to target this class.

4.6 Efficient Interpolation-Based Inference of CDNF Invariants

The goal of this section is to develop an algorithm that can efficiently infer CDNF invariants—invariants that can be rewritten succinctly in both CNF and DNF forms, but are not necessarily monotone or almost-monotone—based on the one-sided fence condition (as opposed to Thm. 4.5.9 which uses the two-sided fence condition). It is inspired by Bshouty's CDNF learning algorithm but significantly diverges from it. We first develop a key technical component of the algorithm, whose goal is to compute monotononizations more efficiently. Then, to develop the intuition of the algorithm's doings, we start by explicitly describing the dual of Alg. 13, and how it correctness stems also from slightly different argument than the one in §4.3.2 (§4.3.3). We build on this intuition to describe our CDNF inference algorithm, using our new efficient monotonization procedure, and prove the algorithm's correctness (§4.6.3).

[7] The candidate in Alg. 6 is an underappxoimation that is increased upon encountering positive examples (much like Algs. 5 and 14), but a negative example (line 5) indicates that the candidate "overshot", whereupon the candidate is subsequently reduced again to an underapproximation.

4.6.1 Super-Efficient Monotonization

In this section we develop an efficient procedure to compute $\mathcal{M}_b(\varphi)$, which is a technical enabler of the results of this section, as well as of §5.6 in Chapter 5. The algorithm, presented in Alg. 15, satisfies the following:

Theorem 4.6.1. *Let φ be a formula and b a cube. The algorithm* MONOTONIZE(φ, b) *computes $\mathcal{M}_b(\varphi)$ in $O(n^2|\mathcal{M}_b(\varphi)|_{\text{dnf}})$ SAT queries and time.*

What distinguishes Thm. 4.6.1 is that the complexity bound depends on the DNF size of the *output*, the monotonization $\mathcal{M}_b(\varphi)$, and *not* on the size of the *input* φ, in contrast to the algorithm by Bshouty [Bsh95] (see Remark 4.6.6).

Algorithm 15 Super-Efficient Monotonization

```
1:  procedure MONOTONIZE(φ, b)
2:      H ← false
3:      while SAT(φ ∧ ¬H) do
4:          let σ_r ⊨ φ ∧ ¬H
5:          v ← GENERALIZE(φ, b, σ_r)
6:          H ← H ∨ cube_b(v)
7:      return H
8:
9:
10:
11:
12: procedure GENERALIZE(φ, b, σ_r)
13:     v ← σ_r; walked ← true
14:     while walked do
15:         walked ← false
16:         for j = 1, ..., n do
17:             if b[p_j] = v[p_j] then
18:                 continue
19:             x ← v[p_j ↦ b[p_j]]
20:             if SAT(φ ∧ ⌈x, π_b(x)⌉) then
21:                 v ← x; walked ← true
22:     return v
```

Starting from the candidate $H = \mathit{false}$, the algorithm iteratively samples—through satisfying models of a SAT query—states that belong in φ but not yet included in H. Every such state σ_r generates a new term in H. Since H is supposed to be b-monotone, the minimal term to include is $\mathit{cube}_b(\sigma_r)$. To be efficient, the algorithm generalizes each example, trying to flip bits to find an example v that also should be included in $\mathcal{M}_b(\varphi)$ and is closer in Hamming distance to b, which would result in a smaller term $\mathit{cube}_b(v)$, thereby including more states in each iteration and converging faster. The criterion for v is that a bit cannot be flipped if this would result in a state x where the Hamming interval $\boxed{x, \pi_b(x)}$ does not intersect φ. Here, $\pi_b(x)$, the projection [e.g. Wie87] of x onto the (possibly partial) cube b is the state s.t.

$$\pi_b(x) = \begin{cases} b[p] & p \in \mathit{dom}(b) \\ x[p] & \text{otherwise} \end{cases},$$

and the Hamming interval $\boxed{\sigma_1, \sigma_2}$ between two states σ_1, σ_2 is the smallest cube that contains both—the conjunction of the literals where these agree. In sum, $\boxed{x, \pi_b(x)}$ is the conjunction of the literals where x, b agree and the literals of x over variables that are not present in b. As we will show, $\boxed{x, \pi_b(x)}$ intersecting with φ is an indicator for x belonging to the monotonization of φ.

The use of SAT queries in the algorithm does not necessarily assume that φ is given explicitly, and indeed in §4.6.3 we apply this algorithm with an implicit representation of φ (using additional copies of the vocabulary).

The rest of this section proves Thm. 4.6.1. First, the result v of generalization is so that when we disjoin the term $cube_b(v)$, we do not "overshoot" to include states that do not belong to the true monotonization:

Lemma 4.6.2. *If $\sigma_r \models \varphi$, then* GENERALIZE(φ, b, σ_r) *returns v s.t. $cube_b(v) \implies \mathcal{M}_b(\varphi)$.*

Proof. v is chosen s.t. $\boxed{v, \pi_b(v)} \cap \varphi \neq \emptyset$—note that this holds trivially in the initial choice of v which is $\sigma_r \models \varphi$. Let $\tilde{\sigma} \models \varphi$ s.t. $\tilde{\sigma} \models \boxed{v, \pi_b(v)}$. The latter means that $\tilde{\sigma} \leq_b v$, because $\boxed{v, \pi_b(v)}$ consists of all the literals in v except for those that disagree with b, so σ agrees with v whenever v, b agree. In more detail, $\tilde{\sigma}$ agrees with v on all $p \notin dom(b)$ (because $\pi_b(v)[p] = v[p]$ on such variables), and for $p \in dom(p)$, if $\tilde{\sigma}[p] \neq v[p]$, if v, b agree on p then likewise $\tilde{\sigma}$ agrees with them (because then $v[p] = \pi_b(v)[p]$ and p is retained in the conjunction that forms the Hamming interval), which satisfies Def. 2.4.2. As also $\tilde{\sigma} \models \varphi$, this implies that $v \models \mathcal{M}_b(\varphi)$ per Def. 2.4.3. Hence $cube_b(v) \implies \mathcal{M}_b(\varphi)$, by Lemma 2.4.9. □

This shows that it is reasonable to disjoin the term $cube_b(v)$ to H in the hope of eventually obtaining $H = \mathcal{M}_b(\varphi)$. The following lemma argues that the algorithm continues to sample states until it converges to the true monotonization.

Lemma 4.6.3. *MONOTONIZE(φ, b) terminates and returns $\mathcal{M}_b(\varphi)$.*

Proof. First we show that when it terminates, the result is correct. Always $H \subseteq \mathcal{M}_b(\varphi)$, because in each iteration we disjoin to H a formula that satisfies the same property, by Lemma 4.6.2. The algorithm terminates when $\varphi \subseteq H$, and H is always a b-monotone formula (by Lemma 2.4.7 the monotonization of H is H itself, which is b-monotone as in Lemma 2.4.4). From the minimality of $\mathcal{M}_b(\varphi)$ (Lemma 2.4.4), necessarily also $\mathcal{M}_b(\varphi) \subseteq H$.

To show termination it suffices to show that H strictly increases in each iteration (because the number of non-equivalent propositional formulas is finite). To see this, note that GENERALIZE(φ, b, σ_r) returns v s.t. $v \leq_b \sigma_r$, since the procedure starts with σ_r and only flips literals to agree with b. This implies that $\sigma_r \models cube_b(v)$, so after the iteration $\sigma_r \models H$ whereas previously $\sigma_r \not\models H$. □

The novelty of the algorithm is its efficiency, which we now turn to establish. The crucial point is the generalization is able to produce, term by term, a minimal representation of $\mathcal{M}_b(\varphi)$. To this end, we first show that $cube_b(v)$ that the algorithm computes in lines 5 to 6 is a *prime implicant* of $\mathcal{M}_b(\varphi)$. Recall that a term t is an *implicant* of a formula ψ if $t \implies \psi$, and it is *prime* if this no longer holds after dropping a literal, that is, for every $\ell \in t$ (as a set of literals), $(\wedge (t \setminus \{\ell\})) \not\implies \psi$. It is *non-trivial* if $t \not\equiv false$ (not an empty set of literals).

Lemma 4.6.4. *If $\sigma_r \models \varphi$, then* GENERALIZE(φ, b, σ_r) *returns v s.t. $cube_b(v)$ is a non-trivial prime implicant of $\mathcal{M}_b(\varphi)$.*

Proof. Lemma 4.6.2 shows that it is an implicant. It is non-trivial because σ_r is a model of it, as shown as part of the proof of Lemma 4.6.3. Suppose that $cube_b(v)$ is not prime. Then there a literal over some variable p that can be dropped. It is present in $cube_b(v)$, which means that $p \in dom(b)$ and $v[p] \neq b[p]$. Then the cube obtained from dropping the literal can be written as $cube_b(x)$ where $x = v[p \mapsto \neg v[p]]$. If this cube is an implicant of $\mathcal{M}_b(\varphi)$, then, because $x \models cube_b(x)$, in particular $x \models \mathcal{M}_b(\varphi)$. By Def. 2.4.3, there is $\tilde{\sigma} \models \varphi$ such that $\tilde{\sigma} \leq_b x$. But the latter implies that $\tilde{\sigma} \in \boxed{x, \pi_b(x)}$, because, by Def. 2.4.2, for every $p \notin dom(b)$, $\tilde{\sigma}[p] = x[p] = \pi_b(x)[p]$ and for every $p \in dom(b)$ where $x, \pi_b(x)$ agree also $\tilde{\sigma}, b$ agree (because $\pi_b(x)[p] = b[p]$). Thus $\boxed{x, \pi_b(x)} \cap \varphi \neq \emptyset$, in contradiction to the choice of v, according to the check in line 20. □

We use the fact that $\mathcal{M}_b(\varphi)$ is b-monotone to show that the monotonization is computed in few iterations:

Lemma 4.6.5. *The number of iterations of the loop in line 3 of* MONOTONIZE(φ, b) *is at most* $|\mathcal{M}_b(\varphi)|_{\text{dnf}}$.

Proof. Lemma 4.6.4 shows that in each iteration we disjoin a prime implicant. It is a property of monotone functions that they have a unique DNF representations with irredundant, which consists of the disjunction of all non-trivial prime implicants [Qui54], and this extends to b-monotone functions (through a simple renaming of variables to make the function monotone). Thus the non-trivial prime implicant we disjoin is a term of the minimal DNF representation of $\mathcal{M}_b(\varphi)$. Each additional σ_r produces a new term, as shown in Lemma 4.6.3. □

We are now ready to prove that the algorithm overall is efficient.

Proof of Thm. 4.6.1. By Lemma 4.6.5 the number of iterations of the loop in line 3 is bounded by $|\mathcal{M}_b(\varphi)|_{\text{dnf}}$. Each iteration calls GENERALIZE, which performs at most n iterations of the loop in line 14 because the same variable is never flipped twice. Each iteration of this loop performs n SAT queries in line 20. Note that the cube $\boxed{x, \pi_b(x)}$ is straightforward to compute in linear time. □

Remark 4.6.6. *Bshouty [Bsh95] used an algorithm for computing $\mathcal{M}_b(\varphi)$ whose complexity is bounded by the DNF input size $|\varphi|_{\text{dnf}}$, whereas Alg. 15's complexity is bounded by the DNF* output *size, $|\mathcal{M}_b(\varphi)|_{\text{dnf}}$, which is never worse (Lemma 2.4.10), and sometimes significantly smaller. When considered as learning algorithms, the improved complexity of Alg. 15 comes at the expense of the need for richer queries: Bshouty's algorithm is similar to Alg. 15 (using an equivalence query in line 3 that produces a positive example—see* GENMQ *of Alg. 6), except that the condition in line 20 is replaced by checking whether $x \models \varphi$. This is a membership query to φ, whereas our check amounts to a disjointness query [Ang87b].*

Algorithm 16 Dual of model-based interpolation-based inference [CIM12, BGKL13]

```
1:  procedure Dual-MB-ITP(Init,δ,Bad,s)
2:      if δ̲^s(Init) ∩ Bad ≠ ∅ then
3:          unsafe
4:      φ ← ¬Bad
5:      while φ not inductive do
6:          let σ,σ' ⊨ φ ∧ δ ∧ ¬φ'
7:          if δ̲^s(Init) ∩ {σ} ≠ ∅ then
8:              restart with larger s
9:          take minimal clause c ⊆ ¬σ s.t. δ̲^s(Init) ⟹ c
10:         φ ← φ ∧ c
11:     return φ
```

Algorithm 17 Interpolation-based inference of CDNF invariants

```
1:  procedure CDNF-ITP(Init,δ,Bad,s)
2:      if δ̲^s(Init) ∩ Bad ≠ ∅ then
3:          unsafe
4:      φ ← ¬Bad
5:      while φ not inductive do
6:          let σ,σ' ⊨ φ ∧ δ ∧ ¬φ'
7:          if δ̲^s(Init) ∩ {σ} ≠ ∅ then
8:              restart with larger s
9:          H ← Monotonize(δ̲^s(Init), σ)
10:         φ ← φ ∧ H
11:     return I
```

4.6.2 Warmup: Dual Interpolation With Clause Minimization

This algorithm computes invariants in CNF and is the dual of Alg. 13. It is guaranteed to converge to an overapproximation of I when I is forwards s-fenced, and converges efficiently when I is a short antimonotone CNF formulas (see Corollary 4.3.9). As usual, the fence condition is necessary to guarantee the success in finding an invariant, but the algorithm is sound regardless, and every inductive invariant it may find is correct even when the fence condition does not hold.

For completeness, the algorithm can be understood without reference to Alg. 13 as follows: After starting the candidate φ as $\neg Bad$, each iteration checks for a counterexample to induction (line 6), whose pre-state σ is excluded from φ at the end of the iteration (line 10). Many states are excluded in each iteration beyond the counterexample, by conjoining to the candidate a minimal *clause* that excludes σ but retains all the states that are reachable in the system in s steps (line 9—this involves up to n queries of s-BMC, each time dropping a literal and checking whether the clause is still valid). If the counterexample cannot be blocked, because it is in fact reachable in s steps, this is an indication that s needs to be larger (line 7) to find a proof or a safety violation. The algorithm detects that the transition system is unsafe in line 3 when s is enough to find an execution from *Init* to *Bad* with at most s transitions.

We prove the following lemma, a counterpart of Lemma 4.3.2, that the algorithm successfully infers an inductive invariant based on the *forwards* fence condition:

Theorem 4.6.7. *Let $(Init, \delta, Bad)$ be a transition system and $s \in \mathbb{N}$. If there exists an inductive invariant I that is* forwards *k-fenced, then* Dual-MB-ITP*$(Init, \delta, Bad, s)$ successfully (albeit potentially in an exponential number of steps) finds an inductive invariant φ such that $I \implies \varphi$.*

However, our proof this time is slightly different, and uses the following result about the monotone hull of a boundary set:

Lemma 4.6.8. *Let I, S, C be sets of states s.t. $\partial^+(I) \subseteq S$ and $C \cap I = \emptyset$. Then $I \subseteq \mathrm{MHull}_C(S)$.*

Proof. Let $\sigma \in I$ and $b \in C$, and show that $\sigma \in \mathcal{M}_b(S)$. To this end, we show that there exists $s \in S$ such that $s \leq_b \sigma$. Consider some shortest path between σ, b in the Hamming cube.

Because $\sigma \models I, b \not\models I$, there is some crossing point $\sigma^+ \in \partial^+(I)$ on that path. Being on the path, it satisfies $\sigma^+ \leq_b \sigma$ (see the "geometric intuition" discussed in §2.4.1). Having $\sigma^+ \in S$ and $\sigma^+ \leq_b \sigma$ implies $\sigma \in \mathcal{M}_b(S)$, concluding the proof. □

Turning back to the correctness of the algorithm, the argument is that always $I \subseteq \mathrm{MHull}_{\mathcal{C}_i}(\mathcal{R}_s) \subseteq \varphi$, where $\mathcal{R}_s$ is the set of s-reachable states $\underline{\delta}^s(\mathit{Init})$, and $\mathcal{C}_i$ is the set of counterexamples σ that the algorithm has encountered so far.

Proof of Thm. 4.6.7. Similar to the overall structure of the proof of Thm. 4.6.7 (but dually), we show that always $I \subseteq \varphi$, which implies that the counterexample σ in line 6 satisfies $\sigma \not\models I$ (because otherwise $\delta(\sigma) \subseteq I \subseteq \varphi$, in contradiction to the choice of σ), and because $\mathcal{R}_s \subseteq I$ this implies that $\sigma \notin \mathcal{R}_s$ and so no restart is required. Because φ strictly decreases in each iteration and the number of non-equivalent formulas is finite, this implies that the algorithm terminates with an inductive invariant. If we denote $\varphi = \psi \wedge \neg \mathit{Bad}$ (that is, ψ is the conjunction learned in iterations of line 5), it suffices to prove that $I \subseteq \psi$, because $I \subseteq \neg \mathit{Bad}$ always holds (as I is an inductive invariant).

The fence condition reads that $\partial^+(I) \subseteq \mathcal{R}_s$. It is easy to see from how the algorithm constructs c that always $\mathcal{R}_s \subseteq \psi$ (initially this holds trivially, and guaranteed afterwards from the condition on c in line 9). It follows that $\partial^+(I) \subseteq \psi$. That always in the algorithm $\sigma \not\models I$ (as explained above) means that $I \cap \mathcal{C}_i = \emptyset$. Hence we can apply Lemma 4.6.8 to obtain that $I \subseteq \mathrm{MHull}_{\mathcal{C}_i}(\psi)$. Since ψ is constructed as a conjunction of clauses that exclude counterexamples in $\mathcal{C}_i$, from Thm. 2.4.19 we have that $\psi \equiv \mathrm{MHull}_{\mathcal{C}_i}(\psi)$, so we have obtained that $I \subseteq \psi$, as desired. □

Towards CDNF inference. According to this proof argument, the important properties of the candidate $\varphi = \psi \wedge \neg \mathit{Bad}$ are that (i) $\mathcal{R}_s \subseteq \psi$, and (ii) $\psi \in \mathrm{MSpan}(\mathcal{C}_i)$ (equivalently, $\psi \equiv \mathrm{MHull}_{\mathcal{C}_i}(\psi)$). These are surprisingly mild requirements; the first can easily be enforced using bounded model checking, and the latter is also in our control, dictating the form of formulas that can be used to exclude the counterexample σ. In Alg. 16 the blocking formula is a clause c, which is σ-monotone, and this is efficient when the k-fenced invariant is antimonotone (Corollary 4.3.9), but might require many iterations for invariants beyond this class. Our CDNF inference algorithm (Alg. 17), presented in the next section, aims to speed convergence by blocking each counterexample using a tighter formula, $\mathcal{M}_\sigma(\mathcal{R}_s)$; this is in fact the tightest possible formula to use while retaining the above two properties.[8]

Although we can query $\mathcal{R}_s$ freely and use Bshouty's algorithm for computing $\mathcal{M}_\sigma(\mathcal{R}_s)$ similarly to Alg. 14, the challenge is that we do not know that $\mathcal{R}_s$ has a short DNF representation, the premise on which this procedure's efficiency rests (see Lemmas 2.4.7 and 4.4.1). The trick is to establish that $\mathcal{M}_\sigma(\mathcal{R}_s) = \mathcal{M}_\sigma(I)$ and find a way to query membership in I sufficiently to

[8] With these blocking formulas, ψ is always $\mathrm{MHull}_{\mathcal{C}_i}(\mathcal{R}_s)$, which is the least overapproximation of $\mathcal{R}_s$ in $\mathrm{MSpan}(\mathcal{C}_i)$ (by Lemma 5.3.1 proven in Chapter 5).

find minimal terms in the DNF representation of I; this is the heart of the technical innovation of the next section.

4.6.3 CDNF Inference With the Fence Condition

We now present our new invariant inference algorithm (Alg. 17), that is guaranteed to run in time polynomial in $n, |I|_{\text{cnf}}, |I|_{\text{dnf}}$ of a fenced invariant I:

Theorem 4.6.9. *Let I be a forwards s-fenced inductive invariant for $(Init, \delta, Bad)$. Then CDNF-ITP$(Init, \delta, Bad, s)$ finds an inductive invariant in at most $|I|_{\text{cnf}} \cdot |I|_{\text{dnf}} \cdot n^2$ of s-BMC checks, $|I|_{\text{cnf}}$ inductiveness checks, and $O(|I|_{\text{cnf}} \cdot |I|_{\text{dnf}} \cdot n^2)$ time.*

Example 4.6.10. *Let I be the set of all states where at least two bits are* 0 and *at least two bits are* 1*. Then $\partial^+(I)$ is the set where exactly two bits are* 0 *(and at least two bits are* 1*) or exactly two bits are* 1 *(and at least two bits are* 0*). Note that $I \setminus \partial^+(I)$ contains many (most) states—those where three or more bits are* 0 *and three or more bits are* 1*. The fence condition requires only from the states in $\partial^+(I)$ to be reachable in s steps.*

I has poly-size representations in both CNF and DNF. We write:

DNF*: there is a choice of four bits with two* 0*'s and two* 1*'s,*

$$I \quad \equiv \quad \bigvee_{1 \le i_1 \ne i_2 \ne i_3 \ne i_4 \le n} (x_{i_1} = 0 \wedge x_{i_2} = 0 \wedge x_{i_3} = 1 \wedge x_{i_4} = 1).$$

CNF*: it is impossible that $n-1$ bits or more are* 1*, likewise for* 0*,*

$$I \quad \equiv \quad \left(\bigwedge_{i=1}^{n} \bigvee_{j \ne i} x_j = 0 \right) \wedge \left(\bigwedge_{i=1}^{n} \bigvee_{j \ne i} x_j = 1 \right).$$

The CNF formula has $2n$ clauses, and the DNF has $\binom{n}{4} = \Theta(n^4)$ terms. Thm. 4.6.9 shows that such I satisfying the fence condition can be inferred in a number of queries and time that is polynomial in n. Note that these formulas fall outside the classes that the previous results of this thesis can handle efficiently (Thm. 4.3.5 and Corollaries 4.3.9, 4.4.3 and 4.4.4) as they are not monotone nor almost-monotone (the number of terms/clauses with negated variables is not constant).

As noted by Bshouty [Bsh95], the class of formulas with short DNF and CNF includes the formulas that can be expressed by a small *decision tree*: a binary tree in which every internal node is labeled by a variable and a leaf by *true*/*false*, and σ satisfies the formula if the path defined by starting from the root, turning left when the σ assigns *false* to the variable labeling the node and right otherwise, reaches a leaf *true*. The size of a decision tree is the number of leaves in the tree. We conclude that:

Corollary 4.6.11. *Let I be a forwards s-fenced inductive invariant for $(Init, \delta, Bad)$, that can be expressed as a decision tree of size m. Then CDNF-ITP$(Init, \delta, Bad, s)$ finds an inductive invariant in at most $m^2 \cdot n^2$ of s-BMC checks, m inductiveness checks, and $O(m^2 \cdot n^2)$ time.*

Proof. As the proof of Corollary 4.5.10 from Thm. 4.5.9. □

Applying the duality of §2.2.1 to the algorithm of Alg. 17 yields exactly the same complexity bounds for CDNF invariants and decision trees under the *backwards* fence condition, instead of the forwards fence condition. The reason is that the CDNF class is closed under negation.

Remark 4.6.12. *Thm. 4.6.9 is not a generalization of Corollary 4.4.3 because some short monotone DNF formulas have large CNF size [MRW05]. However, the algorithm we present in this section can achieve the same result as Alg. 14 if it also uses a known, fix monotone basis instead of discovering the basis on the fly.*

The algorithm CDNF-ITP which attains Thm. 4.6.9 is presented in Alg. 17. Its overall structure is similar to Alg. 16, except the formula used to block a counterexample is the monotonization of the s-reachable states. Specifically, starting from the candidate $\varphi = true$ (line 4, the algorithm iteratively samples counterexamples to induction (line 6) and blocks the pre-state σ from φ by conjoining $\mathcal{M}_\sigma(\underline{\delta}^s(Init))$, computed by invoking Alg. 15. The SAT queries of the form $SAT(\varphi \wedge \theta)$ that Alg. 15 performs (see §4.6.1) have $\varphi = \underline{\delta}^s(Init)$, and they amount to the BMC checks of whether $\underline{\delta}^s(Init) \cap \theta \stackrel{?}{=} \emptyset$.

It is important for the efficiency result that Alg. 17 uses Alg. 15 as a subprocedure. Using Bshouty's procedure (see Remark 4.6.6) would yield a bound of $n \cdot |\underline{\delta}^s(Init)|_{\text{dnf}}$ checks of s-BMC, and it is likely that $\underline{\delta}^s(Init)$ is complex to capture in a formula when s is significant (as common for sets defined by exact reachability, such as the set of the reachable states).

We now proceed to prove the correctness and efficiency of the algorithm (Thm. 4.6.9). Throughout, assume that I is an inductive invariant for $(Init, \delta, Bad)$. I will be s-forwards fenced; we state this explicitly in the premise of lemmas where this assumption is used. The idea behind the correctness and efficiency of Alg. 17 is that $\mathcal{M}_\sigma(I)$ is a stronger formula than the clauses that are produced in Alg. 16, causing the candidate to converge down to the invariant in fewer iterations, while never excluding states that belong to I (because $I \subseteq \mathcal{M}_\sigma(I)$, as used in Lemma 4.6.14). As we will show (in Lemma 4.6.15), this strategy results in a number of iterations that is bounded by the CNF size of I (without further assumptions on the syntactic structure of I). The trick, however, is to show (in Lemma 4.6.13) that what the algorithm computes in line 9 is indeed $\mathcal{M}_\sigma(I)$, even though I is unknown. The crucial observation is that under the fence condition, the monotonization of the s-reachable states matches the monotonization of the invariant (even though these are different sets!). Note that this holds for any invariant that satisfies the fence condition.

We relate the monotonizations of $\underline{\delta}^s(Init), I$ as follows:

Lemma 4.6.13. *If I is forwards s-fenced for $(Init, \delta, Bad)$, and $\sigma \not\models I$, then $\mathcal{M}_\sigma(\underline{\delta}^s(Init)) = \mathcal{M}_\sigma(I)$.*

Proof. Since I is an inductive invariant, $\underline{\delta}^s(\mathit{Init}) \subseteq I$, so $\mathcal{M}_\sigma(\underline{\delta}^s(\mathit{Init})) \subseteq \mathcal{M}_s(I)$ from Lemma 2.4.5. For the other direction we use Lemma 4.6.8: by the fence condition, $\partial^+(I) \subseteq \underline{\delta}^s(\mathit{Init})$ and hence, as $\sigma \not\models I$, we obtain $I \subseteq \mathcal{M}_\sigma(\underline{\delta}^s(\mathit{Init}))$. By Lemma 2.4.4, this implies that $\mathcal{M}_\sigma(I) \subseteq \mathcal{M}_\sigma(\underline{\delta}^s(\mathit{Init}))$. □

We use this to characterize the candidate invariant the algorithm constructs:

Lemma 4.6.14. *If I is forwards s-fenced for $(\mathit{Init}, \delta, \mathit{Bad})$, then in each step of CDNF-ITP$(\mathit{Init}, \delta, \mathit{Bad}, k)$, $\varphi = \mathrm{MHull}_{\mathcal{C}_i}(I) \wedge \neg \mathit{Bad}$, where $\mathcal{C}_i$ is the set of counterexamples σ the algorithm has observed so far. In particular, $I \subseteq \varphi$.*

Proof. First, $I \subseteq \varphi$ holds from the rest of the lemma because $I \subseteq \neg \mathit{Bad}$ (it is an inductive invariant), and $I \subseteq \mathrm{MHull}_{\mathcal{C}_i}(I)$ by Lemma 2.4.14. The proof of $\varphi = \mathrm{MHull}_{\mathcal{C}_i}(I) \wedge \neg \mathit{Bad}$ is by induction on iterations of the loop in line 5. Initially, $\mathcal{C} = \emptyset$ and indeed $\varphi = \neg \mathit{Bad}$. In each iteration, $I \subseteq \varphi$ using the argument above and the induction hypothesis. Hence, the counterexample to induction of line 6 has $\sigma \not\models I$ (otherwise $\sigma' \models I$ because I is an inductive invariant, and this would imply also $\sigma \models \varphi$, in contradiction). Then Lemma 4.6.3 ensures that $H = \mathcal{M}_\sigma(\underline{\delta}^s(\mathit{Init}))$. Lemma 4.6.13 shows that this is $\mathcal{M}_\sigma(I)$, as required. □

Essentially, the algorithm gradually learns a monotone basis (Def. 2.4.17) for I from the counterexamples to induction, and constructs I via the monotone hull w.r.t. this basis. The next lemma shows that the size of the basis that the algorithm finds is bounded by $|I|_{\mathrm{cnf}}$.

Lemma 4.6.15. *If I is forwards s-fenced for $(\mathit{Init}, \delta, \mathit{Bad})$, then CDNF-ITP$(\mathit{Init}, \delta, \mathit{Bad}, k)$ successfully finds an inductive invariant. Further, the number of iterations of the loop in line 5 is at most $|I|_{\mathrm{cnf}}$.*

Proof. Since $\sigma \not\models I$, also it is not a model of the monotonization w.r.t. to itself, $\sigma \not\models \mathcal{M}_\sigma(I)$ (because the only state $x \leq_\sigma \sigma$ is $x = \sigma$—see Defs. 2.4.1 and 2.4.3). This shows, using Lemma 4.6.14, that at least one state is excluded from the candidate φ in each iteration. By the same lemma always $I \implies \varphi$, and the algorithm terminates when φ is inductive, so this shows that the algorithm successfully converges to an inductive invariant.

To see that this occurs in at most $|I|_{\mathrm{cnf}}$ iterations, consider a minimal CNF representation of I, $I = c_1 \wedge \ldots \wedge c_{|I|_{\mathrm{cnf}}}$. We argue that in each iteration produces at least one new clause from that representation, in the sense that for some i, $\varphi \wedge \mathcal{M}_{\sigma_b}(I) \implies c_i$ whereas previously $\varphi \not\Longrightarrow c_i$. Let c_i be the clause that $\sigma \not\models c_i$ (recall e.g. that $\sigma \not\models \varphi$ and $I \subseteq \varphi$). Then $\mathcal{M}_\sigma(I) \subseteq c_i$, since c_i is σ-monotone ($\mathcal{M}_\sigma(c_i) = c_i$, using Lemma 2.4.7, because all the literals disagree with σ) and $I \subseteq c_i$, and $\mathcal{M}_\sigma(I)$ is the smallest such (Lemma 2.4.4). Thus when we conjoin $H = \mathcal{M}_\sigma(I)$ to φ we conjoin at least one new c_i that was not present in a CNF representation of φ; this can happen at most $|I|_{\mathrm{cnf}}$ times. □

Overall:

Proof of Thm. 4.6.9. The algorithm's success in finding an invariant is established in Lemma 4.6.15. As for efficiency, by Lemma 4.6.15, there are at most $|I|_{\text{cnf}}$ iterations of the loop in CDNF-ITP, each performs a single inductiveness query, and calls MONOTONIZE. By Thm. 4.6.1 each such call performs at most $O(n^2|\mathcal{M}_\sigma(I)|_{\text{dnf}})$ s-BMC queries. The claim follows because $|\mathcal{M}_\sigma(I)|_{\text{dnf}} \leq |I|_{\text{dnf}}$ (Lemma 2.4.10). □

Remark 4.6.16 (Comparison to Bshouty's CDNF algorithm)**.** *Our CDNF algorithm, Alg. 17, is inspired by Bshouty's CDNF algorithm (whose code appears in Alg. 6), but diverges from it in several ways. The reason is the different queries available in each setting. Structurally, while the candidate in both algorithms is gradually constructed to be* $\text{MHull}_{\mathcal{C}_i}(I) = \bigwedge_{\sigma\in\mathcal{C}_i} \mathcal{M}_\sigma(I)$ *(*I *being the unknown invariant/formula, and* $\mathcal{C}_i$ *the set of negative examples so far), Alg. 17 constructs each monotonization separately, one by one, whereas Bshouty's algorithm increases all monotonizations simultaneously. Bshouty's design follows from having the source of examples—both positive and negative—equivalence queries, checking whether the candidate matches* I*. A membership query is necessary to decide whether the differentiating example is positive or negative for* I *in order to decide whether to add disjuncts to the existing monotonizations or to add a new monotonization, respectively. This procedure is problematic in invariant inference, because we cannot in general decide, for a counterexample* (σ, σ') *showing that our candidate is not inductive, whether* $\sigma \not\models I$ *(negative) or* $\sigma' \models I$ *(positive) (see §3.6.3). Our solution in §4.5.2 was to assume that the invariant satisfies both the forwards* and *backwards fence condition. Under this assumption it was possible to decide whether* $\sigma \models I$ *for an arbitrary state* σ*. However, this condition is much stronger than a one-sided version of the fence condition. Instead, in our inference algorithm of this section, the candidate is ensured to be an overapproximation of the true* I*, so each counterexample to induction in line 6 yields a negative example. Positive examples are obtained in line 4 from* $\underline{\delta}^s(\mathit{Init}) \subseteq I$*; there is no obvious counterpart to that in exact learning, because in that setting we have no a-priori knowledge of some set* S *that underapproximates* I*, let alone one where we know—as the fence condition guarantees through Lemma 4.6.8—that covering* S *in the monotonization is enough to cover* I*.*

4.7 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 $(\mathit{Init}, \delta, \mathit{Bad})$ is a transition system that admits an inductive invariant I which is backwards or forwards k-fenced (Defs. 4.2.2 and 4.3.8). Suppose that the system is modified in some way to form a new system $(\widehat{\mathit{Init}}, \widehat{\delta}, \widehat{\mathit{Bad}})$ 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 $(\widehat{\mathit{Init}}, \widehat{\delta}, \widehat{\mathit{Bad}})$.

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

4.7.1 Robustness Under Simple Transformations

Isometries

An isometry with respect to the Hamming distance is a permutation of the variables followed by variable translation per §4.4.1 [see e.g. Wie87]. 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 $\widehat{I}$.

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 the boundary of the disjunction/conjunction of invariants is contained in union of the boundaries of the original invariants—as proven in the following lemma— and backwards (forwards) k-reachability is not reduced in either case.

Lemma 4.7.1. *For any two sets of states S_1, S_2, the boundary satisfies (much like in usual topology)*

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

Proof. Let $\sigma^- \in \partial^-(S_1 \cap S_2)$, so wlog. $\sigma^- \not\models S_1$ and it has a Hamming-neighbor $\sigma^+ \models S_1 \cap S_2$. in particular $\sigma^+ \models S_1$ and thus $\sigma^- \in \partial^-(S_1)$. Let $\sigma^- \in \partial^-(S_1 \cup S_2)$, it has a Hamming-neighbor $\sigma^+ \models S_1 \cup S_2$. Assume wlog. that $\sigma^+ \models S_1$. Since $\sigma^- \not\models S_1 \cup S_2$, in particular $\sigma^- \not\models S_1$, and thus $\sigma^- \in \partial^-(S_1)$. The rest of the cases are through the duality $\partial^+(S_1 \cap S_2) = \partial^-(\bar{S}_1 \cup \bar{S}_2) \subseteq \partial^-(\bar{S}_1) \cup \partial^-(\bar{S}_2) = \partial^+(S_1) \cup \partial^+(S_2)$ and $\partial^+(S_1 \cup S_2) = \partial^-(\bar{S}_1 \cap \bar{S}_2) \subseteq \partial^-(\bar{S}_1) \cup \partial^-(\bar{S}_2) = \partial^+(S_1) \cup \partial^+(S_2)$. □

4.7.2 Non-Robustness Under Instrumentation

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 §4.8 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 §4.1.4 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.

Lemma 4.7.2. *Let $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ be obtained as the instrumentation of $(Init, \delta, Bad)$ by the variable q capturing ψ. Let I be over Σ for $(Init, \delta, Bad)$. Then even if I is an inductive invariant, it is* not *forwards k-fenced in $(\widehat{Init}, \widehat{\delta}, \widehat{Bad})$ for any $k \in \mathbb{N}$.*

Proof. Let σ^+ over Σ where $\sigma^+ \in \partial^+(I)$.[9] We extend σ^+ to $\widehat{\sigma^+}$ over $\widehat{\Sigma}$ by interpreting q according to whether $\sigma^+ \models \neg\psi$, that is, q is *true* iff ψ is evaluated to *false* in σ^+. Note that $\widehat{\sigma^+} \models I$ but $\widehat{\sigma^+}$ is "inconsistent", namely, $\widehat{\sigma^+} \not\models q \leftrightarrow \psi$. Therefore, it is unreachable from $\widehat{Init}$ through any number of transitions of $\widehat{\delta}$. It remains to show that $\widehat{\sigma^+} \in \partial^+(I)$. Let σ^- be a Hamming-neighbor of σ^+ such that $\sigma^- \models \neg I$, and extend it to $\widehat{\sigma^-}$ over $\widehat{\Sigma}$ by interpreting q the same way it is interpreted in $\widehat{\sigma^+}$. Then $\widehat{\sigma^+}, \widehat{\sigma^-}$ are Hamming neighbors (w.r.t. $\widehat{\Sigma}$), and $\widehat{\sigma^-} \models \neg I$ as well, since I does not mention q and $\sigma^- \models \neg I$. The claim follows. □

For example, the invariant of Equation (4.1) is forwards 1-fenced before the transformation in §4.1.4 (see Example 4.5.5), but it is no longer forwards k-fenced 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 §4.1.4 demonstrates a similar non-robustness of the backwards fence condition.

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. Thms. 4.3.5 and 4.4.2). Inspired by

[9] The boundaries $\partial^+(I), \partial^-(I)$ are never empty, because $I, \neg I$ are both not empty, and the "crossing point" when walking from one set to the other by gradually flipping bits is in the boundary.

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 antimonotone.

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 antimonotone invariant (to match Corollary 4.3.9). 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 4.7.3. *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 *$k \in \mathbb{N}$.*

Proof. The reasoning is similar in spirit to §4.7.2, but with $\widehat{I}$ which is different from I. As before, a state is called consistent if it indeed interprets q as the negation of the interpretation of p. All reachable states are consistent; we shall show that $\partial^+(\widehat{I})$ includes an inconsistent and hence unreachable state.

First observe that by the construction of I, if $\widehat{\sigma}$ over $\widehat{\Sigma}$ is consistent, then $\pi_\Sigma(\widehat{\sigma}) \models I$ iff $\widehat{\sigma} \models \widehat{I}$.

Denote $\sigma^+ = \sigma$, $\sigma^- = \sigma[p \mapsto true]$, where σ is a state as in the premise of the lemma. Augment them to obtain $\widehat{\sigma^+}, \widehat{\sigma^-}$ by interpreting q to *false* – the value of p in σ^+; note that $\widehat{\sigma^+}$ is inconsistent. In contrast, $\widehat{\sigma^-}$ is consistent, and thus $\sigma^- \models \neg I$, i.e., $\sigma^- \not\models I$, implies that $\widehat{\sigma^-} \not\models \widehat{I}$.

Consider now $\widehat{\sigma^+}$. We know that the consistent $\widehat{\sigma^+}[q \mapsto true]$, which is exactly σ^+ with q interpreted consistently, satisfies I, because σ^+ does. Since it is consistent, $\widehat{\sigma^+}[q \mapsto true] \models \widehat{I}$ as well. But $\widehat{I}$ is antimonotone w.r.t. q, so also $\widehat{\sigma^+} \models \widehat{I}$.

Since both states interpret q the same way, $\widehat{\sigma^+}, \widehat{\sigma^-}$ are Hamming neighbors w.r.t. $\widehat{\Sigma}$. We have shown that $\widehat{\sigma^+} \models I$ whereas $\widehat{\sigma^-} \not\models I$ while $\widehat{\sigma^+}$ is unreachable, and the proof is concluded. □

The case of the backwards fence condition is completely 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 4.7.3) is not in Bad—even though it may reach Bad quickly under δ—because an inconsistent extension of it cannot reach Bad(the system cannot make any step from an inconsistent state) .

4.8 Related Work for Chapter 4

Interpolant generation. Classically, interpolants are generated from unsatisfiability proofs of SAT/SMT solvers, a procedure which has been studied extensively [e.g. McM03, McM05, KW07, CGS10, McM11, VGM15, VNR15]. A relatively recent alternative is to sample states that the interpolant must separate [SNA12]. By iteratively sampling and combining the resulting interpolants, it is possible to compute an interpolant of the (entire) formulas [AM13, DA16, BGKL13, CIM12]. The interpolation procedure in the algorithm we analyze is from Chockler et al. and Bjørner et al. [BGKL13, CIM12], and inspired by IC3/PDR [Bra11, EMB11]. 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 [DA16] is related, with diagrams replacing cubes as the least abstraction of states.

Inference and learning of CDNF and decision trees. Works in learning theory has inspired works on the inference of CDNF formulas and decision trees in various ways. 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 [GLMN14]. We resolve this ambiguity through observations of one-sidedness of the algorithm (§4.5.1) or special invariants to which membership can be implemented (§4.5.2), and also curate a designated inference algorithm for CDNF invariants (§4.6). Garg et al. [GNMR16] target invariants expressed as decision trees over numerical and Boolean attributes, adapting an algorithm by Quinlan [Qui86]. The result in §4.6 applies to decision trees in the propositional setting, and leverages the CDNF algorithm, which admits a bound on the number of hypotheses the learner presents before converging. The CDNF algorithm has been applied by Jung et al. [JKD+15] 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. Our CDNF algorithm from §4.6 is unique in that it is not a direct translation of Bshouty's CDNF algorithm, nor can its structure be explained as a concept learning algorithm. Finally, The CDNF algorithm has also been applied to generate contextual assumptions in assume-guarantee reasoning [CCF+10].

Robustness of program analysis. The drastic consequences of small program changes

on verification tools is sometimes recognized as verification's "butterfly effect" [LP16]. Many program analysis techniques exhibit brittle behaviors [LLFB14, KMW16]. This may be in line with the inherent hardness; for instance, the class of programs for which an abstract interpreter is complete is undecidable [GLR15]. A notable exception is the Houdini algorithm, which efficiently finds the strongest conjunctive invariant without any assumptions on the program [FL01]. In contrast, efficiently inferring invariants from richer syntactic classes is not possible in all programs, because the general case is hard (Thms. 3.2.3 and 3.4.1), 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. FGP16]. 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 [SRW02, FWSS19, SDDA11, BBL+17, Nam07, CBKR19]. Perhaps not surprisingly, there is some empirical evidence that instrumentation can also at times have negative impact on the inference algorithm [FWSS19]. 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 [Bsh18] 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 §4.5.1 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.

Chapter 5

Overapproximation in Property-Directed Reachability From the Monotone Theory

This chapter is based on the results published in [FSSW22, FS22].

The theory of abstract interpretation provides a rich framework to devise invariant inference algorithms. One of the latest breakthroughs in invariant inference is property-directed reachability (PDR), but the research community views PDR and abstract interpretation as mostly unrelated techniques.

This chapter shows that, surprisingly, propositional PDR can be formulated as an abstract interpretation algorithm in a logical domain. More precisely, we define a version of PDR, called Λ-PDR, in which *all* generalizations of counterexamples are used to strengthen a frame. In this way, there is no need to refine frames after their creation, because all the possible supporting facts are included in advance. We analyze this algorithm using notions from Bshouty's monotone theory, originally developed in the context of exact learning. We show that there is an inherent overapproximation between the algorithm's frames that is related to the monotone theory. We then define a new abstract domain in which the best abstract transformer performs this overapproximation, and show that it captures the invariant inference process, i.e., Λ-PDR corresponds to Kleene iterations with the best transformer in this abstract domain. We provide some sufficient conditions for when this process converges in a small number of iterations, with sometimes an exponential gap from the number of iterations required for naive exact forward reachability. Furthermore, we show that under the same conditions, there holds a complexity bound on the number of SAT calls, not just the number of abstract iterations. These results provide a firm theoretical foundation for the benefits of how PDR tackles forward reachability.

5.1 Overview

```
init  x̄ = (x_n, x_{n−1}, ..., x_0) = 0...0,        increase_x():          increase_y():
      ȳ = (y_n, y_{n−1}, ..., y_0) = 0...0,          if z = 0:               if z = 1:
      z = 0                                            x̄ = x̄ + 1 (mod 2^{n+1})    ȳ = ȳ + 1 (mod 2^{n+1})
repeat: increase_x() | increase_y()                    if x̄ = 10...0:
assert ¬(x̄ = 10...0 ∧ ȳ = 11...1 ∧ z = 1)               x̄ = x̄ + 1 (mod 2^{n+1})
```

Figure 5.1: Skip-counter: running example of propositional transition system over the variables $\overline{x} = x_n, \ldots, x_0$. Either `increase_x()` or `increase_y()` is executed in each step according to whether $z = 0$ or $z = 1$, incrementing $\overline{x}$ or $\overline{y}$ resp., but skipping over the value $\overline{x} = 10 \ldots 0$.

5.1.1 PDR, the Frames

How does property-directed reachability find inductive invariants? Given a set of initial states *Init*, a transition relation δ describing one step of the system, and a set of bad states *Bad*, the goal is to find an *inductive invariant*: a formula I such that $\mathit{Init} \Longrightarrow I$, $I \Longrightarrow \neg \mathit{Bad}$, and $\delta(I) \Longrightarrow I$, where the post-image $\delta(X)$ is the set of states that δ reaches in one step from X.[1] Such an I proves *safety*, that no execution of the system can reach a bad state.

The central data structure that PDR uses to find inductive invariants is the *frames*. These are a sequence of formulas $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_N$ that satisfies the following properties, for all $0 \leq i \leq N-1$:

1. $\mathcal{F}_0 = \mathit{Init}$;
2. $\mathcal{F}_i \Longrightarrow \mathcal{F}_{i+1}$;
3. $\delta(\mathcal{F}_i) \Longrightarrow \mathcal{F}_{i+1}$;
4. $\mathcal{F}_i \Longrightarrow \neg \mathit{Bad}$.

In words, the frames start with the set of initial states, grow monotonically, always include the states reachable in one step of δ from the previous frame, and are strong enough to prove safety (except possibly the last frame which is "under construction").

These properties ensure that each $\mathcal{F}_i$ overapproximates the set of states reachable in at most i steps, and yet excludes the bad states; this constitutes a proof of *bounded safety*. However, the ultimate goal of PDR is *un*bounded safety, and it is not clear why frames would avoid "overfitting" to bounded executions, and rather converge to a true inductive invariant. In informal discussions, this is sometimes phrased as the criticism that the algorithm merely "happens to find" a bounded safety proof that generalizes to the unbounded case. Indeed, properties 1–4 of frames do not reflect any bias away from bounded proofs, as they are also satisfied by the exact forward reachability algorithm, $\mathcal{F}_0 = \mathit{Init}, \mathcal{F}_{i+1} = \underline{\delta}(\mathcal{F}_i)$, where $\underline{\delta}(X) = X \cup \delta(X)$ denotes the reflexive closure of the post-image. Exact forward reachability might require many frames to converge to an unbounded proof if some states are reachable only by very long paths.

Consider, for example, the simple family of propositional systems in Fig. 5.1, parametrized by n. A bit z chooses between incrementing a counter $\overline{x}$ or a counter $\overline{y}$, represented in binary

[1] We use a formula and the set of states that satisfy it interchangeably. Unless otherwise stated, the formula to represent a given set of states is chosen arbitrarily.

by $x_n, x_{n-1}, \ldots, x_0$ and $y_n, y_{n-1}, \ldots, y_0$ respectively. The safety property to prove is that it is impossible for $\overline{x}$ to have the value $10\ldots0$ while $\overline{y}$ is $11\ldots1$. This property is not inductive as is (for instance, the state $\overline{x} = 10\ldots0, \overline{y} = 11\ldots10, z = 1$ satisfies the safety property but reaches a bad state in one step); one inductive invariant for this system is

$$\overline{x} \neq 10\ldots0 \wedge \overline{y} = 00\ldots0 \wedge z = 0, \tag{5.1}$$

which implies safety and is closed under a step of the system.

In these systems, exact forward reachability requires $\Omega(2^n)$ iterations before it converges to an inductive invariant. This is because some states, such as $\overline{x} = 10\ldots01, \overline{y} = 00\ldots00, z = 0$, require an exponential number of steps to reach—the system has an exponential *reachability diameter*—so exact forward reachability discovers all reachable states and converges only after that many iterations.

Clearly, invariant inference algorithms must perform some sort of *overapproximation*, or *abstraction*, to overcome this slow convergence. This raises two important questions:

1. What characterizes the abstraction that PDR performs?
2. How does this abstraction achieve faster convergence than exact forward reachability?

The commonly-stated properties of frames do not provide an answer; to address these questions we must dive more deeply into how PDR works.

5.1.2 PDR, the Algorithm

Algorithm 1 PDR [Bra11, EMB11]

```
1:  procedure PDR(Init, δ, Bad)
2:      F_0^pdr ← Init
3:      N ← 0
4:      while ∀1 ≤ i ≤ N. F_i^pdr ⟹̸ F_{i-1}^pdr do
5:          F_{N+1}^pdr ← true
6:          while F_{N+1}^pdr ⟹̸ ¬Bad do
7:              for σ_b ∈ F_{N+1}^pdr ∧ Bad do
8:                  BLOCK(σ_b, N + 1)
9:          N ← N + 1
10:     return F_i^pdr such that F_i^pdr ⟹ F_{i-1}^pdr
11: procedure BLOCK(σ_b, i)
12:     if i = 0 then
13:         unsafe
14:     while δ(F_{i-1}^pdr) ⟹̸ ¬σ_b do
15:         take σ s.t. σ ⊨ F_{i-1}^pdr, (σ, σ_b) ⊨ δ
16:         BLOCK(σ, i − 1)
17:     take c minimal s.t. c ⊆ ¬σ_b and δ(F_{i-1}^pdr) ⟹ c
18:                         and Init ⟹ c
19:     for 1 ≤ j ≤ i do
20:         F_j^pdr ← F_j^pdr ∧ c
```

Alg. 1 presents a simple version of the basic PDR algorithm. The sequence of frames it manipulates are denoted $\mathcal{F}_0^{\text{pdr}}, \ldots, \mathcal{F}_N^{\text{pdr}}$, to distinguish between PDR's frames and frames of other algorithms in this chapter. Initially, $\mathcal{F}_0^{\text{pdr}}$ is initialized to the set of initial states (thereby satisfying property 1). The outer loop terminates once one of the frames is inductive (line 4), which is when $\mathcal{F}_i^{\text{pdr}} \implies \mathcal{F}_{i-1}^{\text{pdr}}$ (because then, from the other properties of frames, $\delta(\mathcal{F}_{i-1}^{\text{pdr}}) \implies \mathcal{F}_i^{\text{pdr}} \implies \mathcal{F}_{i-1}^{\text{pdr}}$). Otherwise, it initializes a new frontier frame to *true* (line 5), and samples bad states (line 7) to block (exclude from the frame) until the frontier frame is strong enough to exclude all bad states (satisfying property 4).

In order to satisfy property 3, before a state σ_b is blocked, the previous frame must be refined it excludes all the pre-states of σ_b (line 14); this is performed by sampling pre-states and blocking them recursively (line 15). Once all the pre-states are blocked in the previous frame, σ_b can be excluded from the current frame. However, at this point, PDR *generalizes* and blocks a *set* of states; this is done by finding *clause* c—also called a *lemma*—that excludes σ_b and still does not exclude any state that is reachable in one step or less from the previous frame (preserving property 3). This is done (in line 18) by starting from all literals (variables or their negations) that are falsified in σ_b, and choosing a subset whose disjunction (which is a clause) satisfies the desired properties. PDR chooses a *minimal subset* in order to exclude as many states as possible. (In practice, this involves a linear number of SAT calls.)[2] The clause is conjoined (in line 20) to the frame as well as the preceding ones (thereby satisfying property 2, relying on $\mathit{Init} \implies c$).

The above is an operational description of how the frames are generated to be overapproximations, but does not lay bare the principles of why they are computed in this way, and how to characterize the abstraction that frames perform.

To study this, we introduce Λ-PDR[3], an alteration of PDR that is simpler for analysis. This algorithm is a theoretical device to study the abstraction in PDR's frames: each frame of Λ-PDR is tighter than the corresponding frame of PDR, and thus ***the overapproximation that Λ-PDR's frames perform is also performed in usual PDR***. We characterize the abstraction that Λ-PDR performs, and show how it can converge more rapidly than exact forward reachability, which sheds light on the abstraction in PDR.

5.1.3 Λ-PDR

The Algorithm

Alg. 18 presents Λ-PDR. Briefly, it constructs frames one after the other, by including all possible lemmas that any execution of PDR might learn; $\mathcal{F}_{i+1}$ is the conjunction of *all clauses* that *block* a state from $\mathcal{B}_k$—the set of states that *can reach a bad state* in at most k steps—yet *retain* the states *reachable in one step* from $\mathcal{F}_i$. The algorithm's essentials are similar to PDR's, with important changes.

First, it is useful for our purpose to decouple two roles that frames serve in PDR. One is as a sequence of approximations to the invariant until the frame where an invariant is found (which is usually somewhere in the middle of the sequence). The other is a way to find counterexamples—which are states that can reach a bad state—without unrolling the transition relation [Bra11]. In Λ-PDR we instead imagine that we are provided, through some arbitrary means (such as unrolling), with $\mathcal{B}_k$, the set of states that can reach a state in *Bad* along some execution of length at most k steps (line 4). This allows us to focus on the other role of frames as approximations

[2] Practical implementations also attempt to push existing lemmas to other frames whenever possible; we omit this for simplicity. (We discuss inductive generalization below, in §5.1.7.)

[3] In homage to Bshouty's Λ-algorithm [Bsh95], not the SARS-CoV-2 variant.

that converge to the invariant. The number k is chosen in advance, independently of the number of frames N.[4]

Second, frames are computed without backtracking to refine previous frames; lemmas to support future frames are learned eagerly, in advance. In particular, convergence is always at the last frame.

Third, whereas PDR "samples" k-backward reachable states and blocks each counterexample in $\mathcal{F}_{i+1}$ using a single, arbitrary (minimal) clause that does not exclude states in $\underline{\delta}(\mathcal{F}_i)$, Λ-PDR generates *all* such clauses—for *any* counterexample state from $\mathcal{B}_k$ (line 11) and *any* order of dropping literals (line 12). This "determinization" makes the algorithm easier to analyze.

Algorithm 18 Λ-PDR

```
1:  procedure Λ-PDR(Init, δ, Bad, k)
2:      F_{-1} ← false
3:      F_0 ← Init
4:      B_k = {σ | δ̲^k(σ) ∩ Bad ≠ ∅}
5:      if Init ∩ B_k ≠ ∅ then unsafe
6:      i ← 0
7:      while F_i ⟹̸ F_{i-1} do
8:          if δ(F_i) ∩ B_k ≠ ∅ then
9:              restart with larger k
10:         F_{i+1} ← true
11:         for σ_b ∈ B_k do
12:             for c ⊆ ¬σ_b do
13:                 if δ̲(F_i) ⟹ c then
14:                     F_{i+1} ← F_{i+1} ∧ c
15:         i ← i + 1
16:     return F_i
```

Overall, the algorithm computes each frame $\mathcal{F}_{i+1}$ iteratively, from the previous $\mathcal{F}_i$, without ever refining previous frames. Each frame is the conjunction of all the clauses that can be obtained as lemmas from blocking any counterexample from $\mathcal{B}_k$ while still overapproximating $\underline{\delta}(\mathcal{F}_i)$. This process continues until an inductive invariant is found (line 7)—unless the current frame does include a counterexample from $\mathcal{B}_k$, which prompts an increase of k (line 8) in order to distinguish between spurious overapproximation and truly unsafe systems (detected in line 5 by an initial state that can reach a bad state in k steps).

As an example, this is how Λ-PDR proceeds on the example of Fig. 5.1 with (say) $k = 1$: The k-backward reachable states $\mathcal{B}_k$ are those where $\overline{x} = 10\ldots00 \wedge y_n y_{n-1}\ldots y_1 = 11\ldots1 \wedge z = 1$ (every value of y_0 yields a backward reachable state). The frame sequence is initialized with $\mathcal{F}_0 = \mathit{Init}$. As $\underline{\delta}(\mathcal{F}_0)$ does not intersect $\mathcal{B}_k$, the algorithm proceeds to computing $\mathcal{F}_1$. It starts as *true*, and the algorithm iterates through the states in $\mathcal{B}_k$ to generate clauses. Suppose that the first counterexample σ_b is $\overline{x} = 10\ldots00 \wedge \overline{y} = 11\ldots10 \wedge z = 1$. We write $\neg\sigma_b = (x_n \neq 1) \vee (x_{n-1} \neq 0) \vee \ldots \vee (x_1 \neq 0) \vee (x_0 \neq 0) \vee (y_n \neq 1) \vee (y_{n-1} \neq 1) \vee \ldots \vee (y_1 \neq 1) \vee (y_0 \neq 0) \vee (z \neq 1)$, and consider every possible sub-clause c thereof, checking whether $\underline{\delta}(\mathcal{F}_0) \implies c$, namely, c includes both $\overline{x} = 00\ldots00, \overline{y} = 00\ldots00, z = 0$ and $\overline{x} = 00\ldots01, \overline{y} = 00\ldots00, z = 0$. In this case, there are several incomparable (and minimal) such c's: $x_n \neq 1$, $y_i \neq 1$ for every $i > 1$, and $z \neq 1$. In Λ-PDR, *all* these potential clauses are conjoined to $\mathcal{F}_1$. The algorithm performs the same procedure for all the counterexamples in $\mathcal{B}_k$. Once this is done, $\mathcal{F}_1$ never changes again in the course of the algorithm, and it becomes the basis for constructing $\mathcal{F}_2$ in the same way, and so on until an inductive invariant is found or a restart becomes necessary. (We later show the resulting $\mathcal{F}_1, \mathcal{F}_2, \ldots$ in this example.)

[4] At first sight $\mathcal{F}_i^{\text{pdr}}$ consists of clauses that exclude counterexamples from a lower backward reachability bound, $\mathcal{B}_{N-i}$; but in fact, pushing lemmas forward means that even $\mathcal{F}_N^{\text{pdr}}$ can include clauses learned at $\mathcal{F}_1^{\text{pdr}}$ from counterexamples in $\mathcal{B}_N$.

PDR Overapproximates at Least as Much as Λ-PDR

The importance of Λ-PDR for our investigation of abstraction in PDR stems from the fact that $\mathcal{F}_i \Longrightarrow \mathcal{F}_i^{\text{pdr}}$, when $k = N$ is the final number of frames in PDR (Corollary 5.8.3). This implies that whatever overapproximation Λ-PDR performs also transfers to PDR: the overapproximation in Λ-PDR is a lower bound for the overapproximation in PDR. The relationship $\mathcal{F}_i \Longrightarrow \mathcal{F}_i^{\text{pdr}}$ holds because every clause c that PDR can use to strengthen $\mathcal{F}_i^{\text{pdr}}$ is used to strengthen $\mathcal{F}_i$ of Λ-PDR (roughly, in PDR, for c to added to $\mathcal{F}_i^{\text{pdr}}$, it must block a counterexample from $\mathcal{B}_k$ and overapproximate the post-image of the previous frame; the same properties would hold for c in Λ-PDR, thus ensuring that c is conjoined to $\mathcal{F}_i$—see Corollary 5.8.3).

Our goal then is to show that Λ-PDR performs significant overapproximation over exact forward reachability, and thereby establish the same for PDR. Our first step is to characterize the overapproximation that Λ-PDR performs, and for this we need tools developed in exact concept learning.

5.1.4 Abstraction from The Monotone Theory

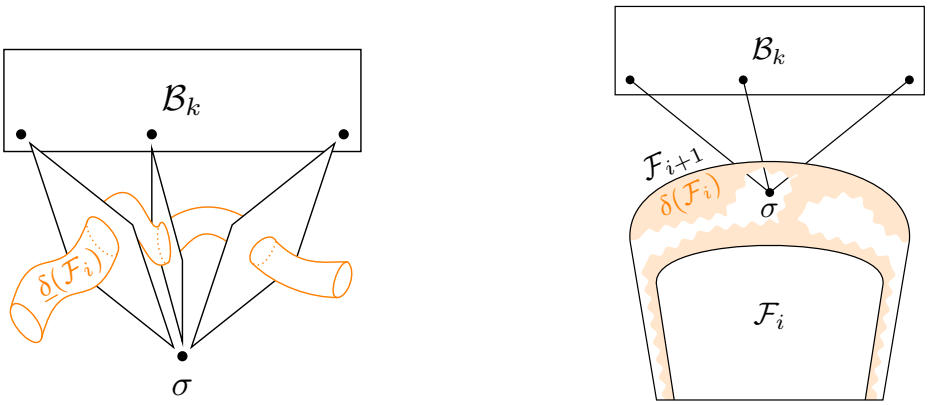
The main technical enabler of our work in this chapter is the observation (Corollary 5.2.2) that in Λ-PDR, there is a well-defined relation between successive frames, through what we call the *monotone hull*:

$$\mathcal{F}_{i+1} = \text{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) \stackrel{\text{def}}{=} \bigwedge_{b \in \mathcal{B}_k} \mathcal{M}_b(\underline{\delta}(\mathcal{F}_i)), \tag{5.2}$$

where $\mathcal{M}_b(\varphi)$ is the central operator in the monotone theory [Bsh95], the *least b-monotone overapproximation of* φ ("b-monotonization" in short). A Boolean function f is b-monotone, when b is a state, if whenever $f(\sigma_1) = 1$ and $\sigma_1 \leq_b \sigma_2$, which means that σ_2 is obtained from σ_1 by flipping bits on which σ_1, b agree, then also $f(\sigma_2) = 1$. $\mathcal{M}_b(\varphi)$ is the *least* b-monotone formula (function) implied by φ. (We elaborate on the technical details in §2.4.) The insight of Equation (5.2) is that, as we show, every lemma in PDR is implied by the monotone hull, and the conjunction of all possible lemmas is exactly the monotone hull. Technically, the observation builds on an equivalent formulation of $\mathcal{M}_b(\varphi)$ in a conjunctive form, which is not explicit in Bshouty's paper (Thm. 5.2.1).

Our central observation is that the monotone hull operator introduces *overapproximation* to the sequence of frames—$\text{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$ can include many more states than $\underline{\delta}(\mathcal{F}_i)$. This is an interesting deviation from Bshouty's use of monotonizations in *exact* learning of an unknown φ, where a set B is chosen such that $\varphi \equiv \text{MHull}_B(\varphi)$; in that case B is said to be a monotone basis for φ, denoted $\varphi \in \text{MSpan}(B)$ (Def. 2.4.18). In contrast, here the monotone hull is applied to intermediate frames, and we are interested in the cases that the set $\mathcal{B}_k$ is *not* a monotone basis for $\underline{\delta}(\mathcal{F}_i)$, for then Equation (5.2) indicates a strict *overapproximation of exact forward reachability* that Λ-PDR performs.

Consider again the running example (Fig. 5.1). One step of exact forward reachability



(a) (b)

Figure 5.2: (a) σ is "protected" by $\underline{\delta}(\mathcal{F}_i)$ from exclusion due to blocking $\mathcal{B}_k$, thus (b) σ is in $\mathcal{F}_{i+1} = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$.

discovers the state $\overline{x} = 00\ldots01, \overline{y} = 00\ldots00, z = 0$ in addition to the initial state. In contrast, by Equation (5.2), the first frame of Λ-PDR with $k = 1$ is $\mathcal{F}_1 = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathit{Init}))$, resulting in

$$\mathcal{F}_1 \;=\; x_n = 0 \wedge \overline{y} = 00\ldots00 \wedge z = 0; \tag{5.3}$$

in a single iteration the algorithm has leaped over an exponential number of steps of δ!

To compute $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathit{Init}))$, in order to arrive at Equation (5.3), we compute the monotone overapproximations. In our example, $\mathcal{B}_k$ is a single cube:

$$\mathcal{B}_k = (x_n = 1 \wedge x_{n-1} = 0 \wedge \ldots \wedge x_1 = 0 \wedge x_0 = 0 \wedge y_n = 1 \wedge \ldots \wedge y_1 = 1 \wedge z = 1),$$

in which case $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_0))$ can be calculated by writing $\underline{\delta}(\mathcal{F}_0)$ in DNF (see Lemmas 2.4.7 and 2.4.13):

$$\begin{aligned}\underline{\delta}(\mathcal{F}_0) = (&x_n = 0 \wedge x_{n-1} = 0 \wedge \ldots \wedge x_1 = 0 \wedge x_0 = 0 \wedge y_n = 0 \wedge \ldots \wedge y_1 = 0 \wedge y_0 = 0 \wedge z = 0)\\ \vee\, (&x_n = 0 \wedge x_{n-1} = 0 \wedge \ldots \wedge x_1 = 0 \wedge x_0 = 1 \wedge y_n = 0 \wedge \ldots \wedge y_1 = 0 \wedge y_0 = 0 \wedge z = 0),\end{aligned}$$

and in each term omitting every literal that agrees with the cube of $\mathcal{B}_k$ (appearing in color). (When $\mathcal{B}_k$ is not a single cube, $\mathcal{F}_{i+1}$ is computed as a conjunction of the above for each cube in $\mathcal{B}_k$.)

What is especially significant about this overapproximation in Λ-PDR is that it exists *in each step*, as we describe in the next subsection (§5.1.5). But before that, we explain the intuition for where this overapproximation stems from.

The cause of overapproximation in Λ-PDR is the special constraints on the lemmas the algorithm can generate. Recall that the states that remain in $\mathcal{F}_{i+1}$ are those that *cannot be excluded* by *any* lemma starting from *any* counterexample, due to the need to satisfy property 3.

Since lemmas are *not* arbitrary formulas, perhaps surprisingly, such states exist beyond the exact post-image. We demonstrate what these states are using the running example. Consider a state σ that satisfies Equation (5.3). Why does no lemma c learned by the algorithm exclude σ, i.e., $\sigma \not\models c$? The reason is that every c excludes some counterexample $b \in \mathcal{B}_k$, and, furthermore, c is a clause—a negation of a cube. A cube is a very rigid geometric shape; if $\neg c$ contains both σ and b then it necessarily contains many other states—it must include all states that are within the smallest cube that contains both σ, b, a.k.a. the Hamming interval between σ, b [e.g. Wie87]. For example, the Hamming interval between $\sigma = (\overline{x} = 011\ldots101, \overline{y} = 00\ldots00, z = 0)$ and $b = (\overline{x} = 10\ldots00, \overline{y} = 11\ldots10, z = 1)$ is $x_1 = 0 \wedge y_0 = 0$—the conjunction of the literals (or constraints) that hold in both σ, b. However, $\neg c$ must not contain any state in $\underline{\delta}(\mathcal{F}_i)$, so the Hamming interval between σ, b cannot intersect $\underline{\delta}(\mathcal{F}_i)$. In our example, the Hamming interval between σ and b includes the state $\widetilde{\sigma} = (\overline{x} = 00\ldots00, \overline{y} = 00\ldots00, z = 0)$, and $\widetilde{\sigma} \in \underline{\delta}(\mathcal{F}_0)$, so a lemma c that excludes σ and originates from blocking b cannot be conjoined to $\mathcal{F}_1$.

Put differently, $\widetilde{\sigma} \in \underline{\delta}(\mathcal{F}_0)$ "protects" $\sigma \in \mathcal{F}_1$ from being excluded due to b. In general, a state σ is included in $\mathcal{F}_{i+1}$ if a protector state $\widetilde{\sigma} \in \mathcal{F}_i$ exists *for every* $b \in \mathcal{B}_k$, namely, the Hamming interval between σ, b crosses $\underline{\delta}(\mathcal{F}_i)$ for all b's (Fig. 5.2). In our example, the same $\widetilde{\sigma}$ actually protects every $\sigma \in \mathcal{F}_1$ from exclusion due to any $b \in \mathcal{B}_k$, but multiple protector states may be necessary in general.

The idea of protector states explains why $\mathcal{F}_{i+1}$ is $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$ (Equation (5.2)). Every state $\widetilde{\sigma} \in \underline{\delta}(\mathcal{F}_0)$ is a protector state. The states that $\widetilde{\sigma}$ protects from b are the states σ such that $\widetilde{\sigma}$ is in the Hamming interval between σ and b; these states are "farther away" from b than $\widetilde{\sigma}$, in the sense of $\widetilde{\sigma} \leq_b \sigma$ as defined above (and formally in Def. 2.4.1). The set protected from b by $\underline{\delta}(\mathcal{F}_i)$ is therefore $\mathcal{M}_b(\underline{\delta}(\mathcal{F}_i))$, and the states that are protected from all $b \in \mathcal{B}_k$ are the conjunction over all b's, namely $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$.

5.1.5 Successive Overapproximation: Abstract Interpretation

The overapproximation of Equation (5.2) is present between each two consecutive frames; it is thus performed repeatedly, using the previous overapproximation as the starting point of the next. In §5.3 we show that Λ-PDR can be cast as ***abstract interpretation in a new logical domain***, of the formulas in $\mathrm{MSpan}(\mathcal{B}_k)$, the formulas φ s.t. $\mathrm{MHull}_{\mathcal{B}_k}(\varphi) \equiv \varphi$, which are the formulas expressible by a conjunction of clauses that each excludes a state from $\mathcal{B}_k$ (Def. 2.4.18). ***The frames of Λ-PDR are completely characterized as Kleene iterations with the best abstract transformer*** in this domain (Thm. 5.3.5), when correcting for the slightly different initial frame ($\mathcal{F}_0 = \mathit{Init}$ vs. $\mathrm{MHull}_{\mathcal{B}_k}(\mathit{Init})$; we show that the resulting difference in the number of frames is at most one).

Repeatedly applying the abstraction can cause Λ-PDR to converge much faster than exact forward reachability (illustrated schematically in Fig. 5.3). For example, the frames of Λ-PDR on the running example are displayed in Fig. 5.4 (we perform the calculation in detail in Example 5.2.3), and $\mathcal{F}_3$ is none other than the inductive invariant of Equation (5.1). In this

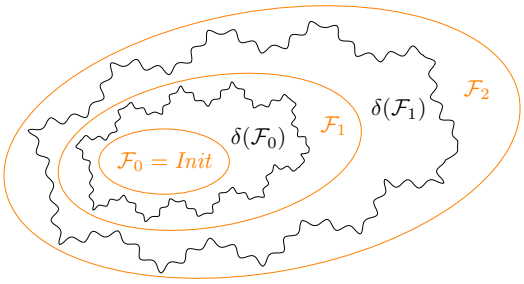


Figure 5.3: Repeatedly interleaving the post-image and monotone hull operators results in successive overapproximation.

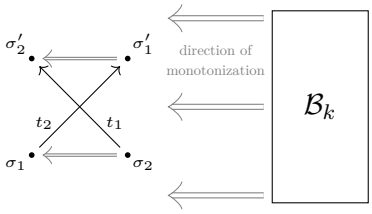


Figure 5.5: The monotonization of a transition $t_1 = (\sigma_1, \sigma'_1) \in \delta$ subsumes that of $t_2 = (\sigma_2, \sigma'_2) \in \delta$ if σ_1 is farther from the backward reachable cube than σ_2, and σ'_1 is closer than σ'_2. If few transitions subsume the rest, $|\delta^{\mathbf{x}}|_{\text{dnf}}$ is small, which implies convergence with few frames for Λ-PDR.

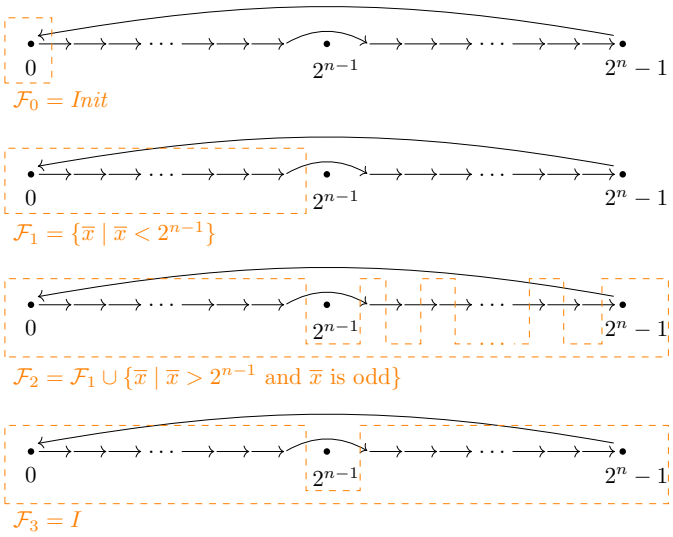


Figure 5.4: The frames of Λ-PDR on the running example (only values of $\overline{x}$ are displayed, always $\overline{y} = 0 \ldots 0, z = 0$). The number of frames required for convergence is always 4, independent of the parameter n.

way, ***successive overapproximation can lead to convergence in a smaller number of frames than exact forward reachability***; in this example, Λ-PDR converges in 4 frames, rather than an exponential number as exact forward reachability would use.[5] In §5.7, we show that Λ-PDR holds a similar advantage over the unrolling depth of an interpolation-based algorithm.

5.1.6 Convergence Bounds via (Hyper)Diameter Bounds

When does the successive overapproximation of Λ-PDR terminate in a small number of iterations? The lattice height of the abstract domain is exponential in the number of variables (see §5.3), and rapid convergence depends on properties of the transition system rather than of the abstract domain.

We prove a convergence bound using one such property: ***when the DNF size*** (the number of terms in the smallest DNF representation) ***of specific monotone overapproximations of the transition relation are small, then Λ-PDR terminates after a small number of iterations*** (Thms. 5.4.2 and 5.5.2). The central idea is to relate Λ-PDR to ***exact forward reachability in an "abstract" transition system***, and then bound the number of iterations using techniques for analyzing the diameter of transition systems.

[5] The operations in the abstract domain are not efficient; we focus on the number of iterations until convergence.

Let us consider an example where Thm. 5.4.2 derives an efficient convergence bound (specifically, a linear bound, as opposed to the potential exponential number of iterations) through a syntactic analysis of the transition system in question. Consider the same example of Fig. 5.1, but with additional transitions that "bounce back" from $\overline{x} = 01\ldots11$ to $\overline{x} = 00\ldots00$ and from $\overline{x} = 11\ldots11$ to any $\overline{x} = 10\ldots010\ldots0$ (number with msb and other variable 1). (The new transitions have no effect on the behavior of either exact forward reachability or Λ-PDR;[6] we explain below why this is needed to obtain a good bound through Thm. 5.4.2.)

In this transition system, we bound the number of iterations of Λ-PDR by applying Thm. 5.4.2 to the part of δ restricted to states where $\overline{y} = 0\ldots0, z = 0$ (this is valid because both the transitions of δ and monotonization w.r.t. $\mathcal{B}_k$, $\mathcal{M}_{\overline{x}=10\ldots0,\overline{y}=1\ldots1,z=1}(\cdot)$, leave $\overline{y} = 0\ldots0, z = 0$ unchanged—see Remark 5.5.5). The transition relation $\tilde{\delta} = \delta|_{\overline{y}\leftarrow 00\ldots00, z\leftarrow 0}$ can be written in DNF (as a double-vocabulary formula, with unprimed variables for the pre-state and primed variables for the post-state) as the disjunction of individual transitions, as appears on the left hand side here (colors and boxes should be ignored at this point):

$$\tilde{\delta} = (\overline{x} = 000\ldots0000 \wedge \overline{x}' = 000\ldots0001) \qquad \delta^{\mathbf{x}} = \mathcal{M}_{\overline{x}=011\ldots11,\overline{x}'=100\ldots00}(\tilde{\delta}) = \tag{5.4}$$

$$\vee\ (\overline{x} = 000\ldots0001 \wedge \overline{x}' = 000\ldots0010) \tag{5.5}$$

$$\vee\ (\overline{x} = 000\ldots0010 \wedge \overline{x}' = 000\ldots0011) \tag{5.6}$$

$$\vee\ (\overline{x} = 000\ldots0011 \wedge \overline{x}' = 000\ldots0100) \tag{5.7}$$

$$\vee\ \ldots \tag{5.8}$$

$$\vee\ \boxed{(\overline{x} = 011\ldots1111 \wedge \overline{x}' = 000\ldots0000)} \qquad \vee\ x'_n = 0 \tag{5.9}$$

$$\vee\ \boxed{(\overline{x} = 011\ldots1111 \wedge \overline{x}' = 100\ldots0001)} \qquad \vee\ x'_0 = 1 \tag{5.10}$$

$$\vee\ (\overline{x} = 100\ldots0001 \wedge \overline{x}' = 100\ldots0010) \tag{5.11}$$

$$\vee\ (\overline{x} = 100\ldots0010 \wedge \overline{x}' = 100\ldots0011) \tag{5.12}$$

$$\vee\ \ldots \tag{5.13}$$

$$\vee\ (\overline{x} = 111\ldots1110 \wedge \overline{x}' = 111\ldots1111) \tag{5.14}$$

$$\vee\ (\overline{x} = 111\ldots1111 \wedge \overline{x}' = 000\ldots0000) \tag{5.15}$$

$$\vee\ (\overline{x} = 111\ldots1111 \wedge \overline{x}' = 100\ldots0001) \tag{5.16}$$

$$\vee\ \boxed{(\overline{x} = 111\ldots1111 \wedge \overline{x}' = 100\ldots0010)} \qquad \vee\ (x_n = 1 \wedge x'_1 = 1) \tag{5.17}$$

$$\vee\ \boxed{(\overline{x} = 111\ldots1111 \wedge \overline{x}' = 100\ldots0100)} \qquad \vee\ (x_n = 1 \wedge x'_2 = 1) \tag{5.18}$$

$$\vee\ \boxed{(\overline{x} = 111\ldots1111 \wedge \overline{x}' = 100\ldots1000)} \qquad \vee\ (x_n = 1 \wedge x'_3 = 1) \tag{5.19}$$

$$\vee\ \boxed{\cdots} \qquad \vee\ \ldots \tag{5.20}$$

To compute a bound for the number of frames in Λ-PDR, we perform a monotonization of the

[6] The algorithms are affected by the new transitions when they arrive to the transitions' pre-states, but at this point both algorithms will have already arrived to the smaller numbers in the post-states of the new transitions, resulting in the same frames as without these transitions.

(two-vocabulary) transition relation. Recall that in this example $\mathcal{B}_k$ consists of a single cube, in which case we need only consider one monotonization (the case of more complex syntactic forms of $\mathcal{B}_k$ is discussed later): examine the monotonization that omits literals that agree with $\mathcal{B}_k$ in the post-state, and *conversely* in the pre-state, $\delta^{\times} = \mathcal{M}_{\overline{x}=01\ldots11,\overline{x}'=10\ldots00}(\tilde{\delta})$. The literals in $\tilde{\delta}$ that are omitted in $\delta^{\times}$ appear colored. As we show in Lemma 5.4.7, the resulting transition relation captures the behavior of Λ-PDR: the set of i-reachable states of $\delta^{\times}$ matches the i'th frame of the Kleene iterations with the best transformer for δ in the MSpan($\mathcal{B}_k$) domain. Hence, ***bounds on the diameter of the abstract system result in bounds on the number of frames*** of Λ-PDR.

To bound the diameter, we consider the DNF size $\delta^{\times}$. The monotonization term-by-term from $\tilde{\delta}$ creates many redundant terms; the terms that originate from the transitions marked by boxes above subsume all the others.[7] This generates a DNF representation of $\delta^{\times}$ with linear number of terms (appearing in the right-hand side above)—even though the original $\tilde{\delta}$ has an exponential number of terms in its DNF representation. By Thm. 5.4.2 we deduce from the linear DNF size of $\delta^{\times}$ that Λ-PDR converges in at most a linear number of frames.

One way to think about the difference between $\delta, \delta^{\times}$ is by the way transitions in δ give rise to the transitions in $\delta^{\times}$, illustrated in Fig. 5.5. A transition of $\delta^{\times}$ can *abstract* and move away from $\mathcal{B}_k$, then follow a *concrete* transition of δ, and from the resulting post-state again *abstract* and move in the direction away from $\mathcal{B}_k$. In this way, it may be possible for $\delta^{\times}$ to use the transition (σ_1, σ_1') in order to arrive from σ_2 to σ_2', even if (σ_2, σ_2') were not a transition of δ (see Fig. 5.5). When this is the case for $(\sigma_1, \sigma_1'), (\sigma_2, \sigma_2') \in \delta$, the transitions of $\delta^{\times}$ that use the concrete transition (σ_2, σ_2') are also possible using the concrete transition (σ_1, σ_1'); hence the term generated from (σ_2, σ_2') can be discarded in the monotonization of δ to obtain $\delta^{\times}$, because it is subsumed by the term generated from (σ_1, σ_1'). Roughly, Thm. 5.4.2 shows that Λ-PDR converges rapidly whenever there is a small number of transitions that subsume the others, by going from a pre-state σ that is "very far" from $\mathcal{B}_k$ in Hamming distance compared to the pre-states of other transitions, to the post-state σ' that is "very close" to $\mathcal{B}_k$ compared to the post-states of other transitions. This is an intuition for how a small $|\delta^{\times}|_{\text{dnf}}$ can arise from the monotonization of the fully-expanded DNF representation of δ. (The starting point for monotonization can also be a more succinct DNF representation of δ, in which case the intuition for an even shorter DNF representation of $\delta^{\times}$ is similar.)

If we were not to add the "bounce back" transitions to the example of Fig. 5.1, still monotonization of the transition relation produces an abstract transition system whose reachable states coincide with Λ-PDR's frames, and whose diameter corresponds to the number of iterations

[7] The term arising from the "bounce back" transition with msb 0 in Equation (5.9) subsumes all other terms that originate from transitions where the msb is 0 in both the pre-state and the post-state (Equations (5.4) to (5.8)), as well as the term originating from the "wraparound" transition in Equation (5.15); the term arising from the "skip" transition in Equation (5.10) subsumes the term originating from the transition in Equation (5.16); and the terms arising from "bounce back" transitions with msb 1 in Equations (5.16) to (5.20) (including Equation (5.16) which is subsumed by Equation (5.10)) subsume all the terms that arise from transitions where the msb is 1 both in the pre-state and the post-state (Equations (5.11) to (5.14)).

in which Λ-PDR converges. However, in that case the bound of Thm. 5.4.2 is poor, because in this case the DNF-size is a poor estimate of the abstract system's diameter (see Example 5.4.10).

For when $\mathcal{B}_k$ consists of multiple cubes, we generalize Thm. 5.4.2 to Thm. 5.5.2, bounding the number of frames by a product of monotonizations of δ and *Init*. In the proof, the construction involves not an ordinary transition system, but a *hyper*transition system: the hypertransition relation $\delta^{\times}$ arrives through concrete transitions to a *set* of states, and abstracts from them to a state "protected" by that set, because the abstraction requires a "protector" state from every state in $\mathcal{B}_k$ (see Fig. 5.2). A similar diameter bound using the DNF size of the hypertransition relation $\delta^{\times}$ applies. We show that $\delta^{\times}$ can be written as a conjunction of per-cube monotonizations, leading to a bound by the product of DNF sizes of monotonizations of δ and *Init* (see §5.5).

This technique does not explain rapid convergence of Λ-PDR in full generality, but provides one explanation for how the abstraction can bring this about.

5.1.7 PDR, Revisited

Through Λ-PDR, we have shown how PDR's frames perform an abstract interpretation in a domain founded on the monotone theory, and how such an abstraction can lead to faster convergence. We observe that these important characteristics of PDR are concealed in a simple property of PDR's frames: that they can be written in CNF so that every clause excludes at least one state from $\mathcal{B}_N$ (Lemma 5.8.2). In the monotone theory from above this reads that for every $1 \leq i \leq N$,

5. $\mathcal{F}_i \in \mathrm{MSpan}(\mathcal{B}_N)$.

The frames of Λ-PDR are the least set of states that satisfy this property together with properties 1–4 from §5.1.1 (Lemma 5.8.1), and the frames of PDR overapproximate them (Corollary 5.8.3). Property 5 is the regularization in our abstract domain (§5.3), and we have shown that it can lead to faster convergence than exact post-image computations—although PDR does not necessarily converge in the same number of frames as Λ-PDR, due to its additional overapproximation and heuristics.

The fact that PDR's frames are not the least to satisfy the above properties can have some benefits. We show two:

- **Faster convergence**: In some cases Λ-PDR performs little or no abstraction over exact forward reachability, but the fact that PDR only samples a subset of the possible lemmas can guarantee fast convergence. We show an example where Λ-PDR requires an exponential number of frames, whereas a linear number always suffices for standard PDR.

- **Frame size**: Λ-PDR's frames may be (needlessly) complex to represent as a formula. We show an example where some frames of Λ-PDR necessarily have an exponential DNF or

CNF size, whereas standard PDR can converge in the same number of frames that include only a small number of important lemmas.

Discussion: Additional PDR Features

Our study focuses on what is, in our view, the most basic version of PDR. Our approach provides an interesting starting point to a discussion of two common, more advanced features of PDR.

Other forms of generalization. Inductive generalization [Bra11] minimizes lemmas using a stronger criterion: a lemma c can be learned in $\mathcal{F}_{i+1}^{\text{pdr}}$ if it is inductive *relative* to $\mathcal{F}_i$—whether $\underline{\delta}(\mathcal{F}_i^{\text{pdr}} \wedge c) \implies c$, namely, checking whether c holds in the post-state while also assuming c in the pre-state. At first sight, this feature is not present in Λ-PDR, which uses the standard check (Alg. 18, line 13). Surprisingly, lemmas that PDR can generate using inductive generalization are also present in Λ-PDR (with $k = N$). This is a consequence of the fact that PDR with inductive generalization still satisfies properties 1–4 [Bra11, EMB11], as well as property 5. The optimization of inductive generalization becomes important only when lazily backtracking to refine previous frames. Other techniques, such as ternary simulation [EMB11], propagate sets of states to block together. If any of the states is in $\mathcal{B}_k$, the resulting lemma is present also in Λ-PDR.

May-counterexamples. Some variants of PDR produce lemmas by blocking may counterexamples [GI15] that are not necessarily backward reachable, as a way to encourage pushing existing lemmas to later frames. In Λ-PDR, all admissible lemmas from $\text{MSpan}(\mathcal{B}_k)$ are always included, hence may counterexamples are not useful for pushing such lemmas. However, may counterexamples also mean that lemmas no longer necessarily block states in $\mathcal{B}_N$, which could be beneficial if a large N is required to have an inductive invariant $I \in \text{MSpan}(\mathcal{B}_N)$. (It is a necessary condition PDR; in Λ-PDR it is both necessary and sufficient, see Corollary 5.3.7.) This could be thought of as (heuristically) increasing the set $\mathcal{B}_k$. In Λ-PDR, this results in a richer abstract domain that includes more inductive invariants but leads to tighter frames with less overapproximation. The theoretical ramifications of this beyond Λ-PDR merit more study.

Notation. For brevity, we denote $\left(\underline{\delta^{-1}}\right)^k (\mathit{Bad})$ by $\mathcal{B}_k$. In this chapter, the distinction between syntax and semantics is of lesser importance (because the results are not concerned with an efficient representation of formulas). Henceforth, we identify a formula with the set of its valuations, and a set of valuations with an arbitrary formula that represents it, chosen arbitrarily (which always exists in propositional logic). We further shorten, and identify $\mathit{cube}(\sigma)$ with σ (since $\mathit{cube}(\sigma)$ represents the singelton $\{\sigma\}$).

5.2 The Monotone Theory for Λ-PDR

In this section we recall the monotone theory by Bshouty [Bsh95], extend it with a conjunctive representation of the least b-monotone overapproximation, and use it to derive the relation between successive frames in Λ-PDR (Equation (5.2)).

Recall (from §2.4) that given a cube b, a formula ψ is b-monotone if it is closed under flipping literals to disagree with b (Def. 2.4.2). The least b-monotone overapproximation $\mathcal{M}_b(\varphi)$ of a formula φ is the formula that contains φ and all the states σ from which there is a shortest path between σ, b that crosses φ (Def. 2.4.3). Given a set of states B, the monotone hull $\mathrm{MHull}_B(\varphi)$ is the conjunction of per-cube monotone overapproximations, $\mathcal{M}_{b_1}(\varphi) \wedge \ldots \wedge \mathcal{M}_{b_m}(\varphi)$ where $B = b_1 \vee \ldots \vee b_m$ (Def. 2.4.12, Lemma 2.4.13).

The importance of b-monotonicity for us is that if $\sigma \models b$, then, as we shall see (in Thm. 5.2.1), any clause $c \subseteq \neg\sigma$ is a b-monotone formula. The reason is that if $v \models c$ and we flip variables to disagree more with b, we can only make v satisfy more literals of c than before: all the variables in b appear in the opposite polarity in $\neg\sigma$ and hence also in c, so flipping them in $v \models c$ to disagree with b makes them agree with c even more, and the result also satisfies c.

Further, the lemmas that PDR learns are not just clauses, but clauses that overapproximate the post-image of the previous frame, hence they are b-monotone overapproximations of that set. There are many b-monotone overapproximations, but for our purposes $\mathcal{M}_b(\varphi)$ is instrumental, in ways that are quite different from its original use in learning theory (discussed in §2.4.4). We show that $\mathcal{M}_b(\varphi)$ is exactly the conjunction of all the clauses c that overapproximate φ and can arise from blocking a state in b (Thm. 5.2.1), matching generalization in the construction of frames of Λ-PDR (Corollary 5.8.3).

Our main technical observation, connecting the monotone hull to Λ-PDR, is that the monotone hull has an equivalent CNF form, as the conjunction of all overapproxmating clauses that exclude a state in B:

Theorem 5.2.1. $\mathrm{MHull}_B(\varphi) \equiv \bigwedge \{c \mid c \textit{ is a clause, } \varphi \Longrightarrow c, \textit{ and } \exists b \in B.\, b \not\models c\}$.

Proof. $\Longrightarrow$: Let c be a clause as in the rhs. Then there exists $b \in B$ s.t. $b \not\models c$. It suffices to show that $\mathcal{M}_b(\varphi) \Longrightarrow c$. Recall that c is a disjunction of literals; since $b \not\models c$, all those literals are falsified in b. Hence, by Lemma 2.4.7, $\mathcal{M}_b(c) \equiv c$. Now $\varphi \Longrightarrow c$ (by the choice of c), and Lemma 2.4.5 yields $\mathcal{M}_b(\varphi) \Longrightarrow \mathcal{M}_b(c) \equiv c$.
$\Longleftarrow$: Let σ be a model of the rhs. We want to prove that $\sigma \models \mathcal{M}_b(\varphi)$ for every $b \in B$. Assume otherwise. Take d as the conjunction of all literals that hold in both σ and b (if this set is empty, $d = \textit{true}$). Clearly d is a term and $b, \sigma \models d$. Take $c = \neg d$; then c is a clause and $b \not\models c$. It remains to show that $\varphi \Longrightarrow c$, because then c belongs to the rhs but $\sigma \not\models c$, in contradiction to the premise. To see this, note that $c = \mathcal{M}_b(\neg\sigma)$ by Lemma 2.4.7—all the literals from the clause $\neg\sigma$ on which b disagrees (because b agrees on them with σ). $\varphi \Longrightarrow \neg\sigma$ because $\sigma \not\models \varphi$ (as $\sigma \not\models \mathcal{M}_b(\varphi)$, which is an overapproximation of φ). Hence Lemma 2.4.5 implies $\mathcal{M}_b(\varphi) \Longrightarrow \mathcal{M}_b(\neg\sigma) = c$, and in particular $\varphi \Longrightarrow c$. The claim follows. □

We can now derive Equation (5.2):

Corollary 5.2.2. *In Λ-PDR (Alg. 18),* $\mathcal{F}_{i+1} = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$.

Proof. $\mathcal{F}_{i+1}$ is the conjunction formed by the process that for each $\sigma_b \in \mathcal{B}_k$ (line 11 of Alg. 18) iterates in line 12 over all clauses that exclude σ_b, and conjoins c if it overapproximates $\underline{\delta}(\mathcal{F}_i)$ (line 13). By Thm. 5.2.1 this is $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$. □

Example 5.2.3. *The above lemmas are the basis for our presentation in §5.1.4 of $\mathcal{F}_1$ on Fig. 5.1 as Equation (5.3). Let us now use these lemmas to describe later frames in that execution. Recall that $\mathcal{B}_k$ is the cube $\overline{x} = 10\ldots0 \wedge y_n y_{n-1}\ldots y_1 = 11\ldots1 \wedge z = 1$, denote it by b. For the next frame, $\underline{\delta}(\mathcal{F}_1) = \mathcal{F}_1 \vee (\overline{x} = 10\ldots01 \wedge \overline{y} = 0\ldots0 \wedge z = 0)$. Then*

$$\begin{aligned}\mathcal{F}_2 &\underset{\text{Thm. 5.2.1}}{=} \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_1)) \underset{\text{Lemma 2.4.13}}{=} \mathcal{M}_b(\underline{\delta}(\mathcal{F}_1)) \\ &\underset{\text{Lemma 2.4.7}}{=} \mathcal{M}_b(\mathcal{F}_1) \vee \mathcal{M}_b(\overline{x} = 10\ldots01 \wedge \overline{y} = 0\ldots0 \wedge z = 0) \\ &\underset{\text{Lemma 2.4.7}}{=} \mathcal{F}_1 \vee (x_0 = 1 \wedge \overline{y} = 0\ldots0 \wedge z = 0).\end{aligned}$$

For the next frame, $\underline{\delta}(\mathcal{F}_2) = \mathcal{F}_2 \vee (x_0 = 0 \wedge \overline{y} = 0\ldots0 \wedge z = 0) \wedge \neg(\overline{x} = 1\ldots0 \wedge \overline{y} = 0\ldots0 \wedge z = 0)$. This is equivalent to $(\overline{y} = 0\ldots0 \wedge z = 0) \wedge \neg(\overline{x} = 1\ldots0 \wedge \overline{y} = 0\ldots0 \wedge z = 0)$, which is already b-monotone and hence this is also $\mathcal{F}_3 = \mathrm{MHull}_b(\underline{\delta}(\mathcal{F}_2))$. $\underline{\delta}(\mathcal{F}_3) = \mathcal{F}_3$, and so $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_3)) = \mathcal{F}_3$ by Lemma 2.4.16, and the algorithm converges.

Example 5.2.4. *In the previous example, $\mathcal{B}_k$ consisted of a single cube. To exemplify the more general case, consider a system over n variables $x_1, \ldots, x_n$, with $Init = (x_1 = \ldots = x_n = 0)$, Bad the set of states with exactly one variable 1, and δ that non-deterministically chooses some $i \neq j$ with $x_i = x_j = 0$ and sets $x_i \leftarrow 1$ and $x_j \leftarrow 1$. Take $k = 0$. Then $\mathcal{B}_k = b_1 \vee \ldots \vee b_n$ where b_i is $x_i = 1 \wedge \bigwedge_{j \neq i}(x_j = 0)$. After one step, $\underline{\delta}(\mathcal{F}_0) = (0\ldots0) \vee \bigvee_{i_1 \neq i_2}(x_{i_1} = x_{i_2} = 1 \wedge \bigwedge_{j \notin \{i_1, i_2\}}(x_j = 0))$ is the set of states where there are zero or two variables 1. Then for every b_i,*

$$\mathcal{M}_{b_i}(\underline{\delta}(\mathcal{F}_0)) \underset{\text{Lemma 2.4.7}}{=} (x_i = 1) \vee \left(\bigvee_{i_1 \neq i_2, i \notin \{i_1, i_2\}} x_{i_1} = x_{i_2} = x_i = 1\right) \vee \left(\bigvee_{i_1 \neq i_2, i = i_1} x_{i_2} = 1\right) = \bigvee_j (x_j = 1).$$

Hence, $\mathcal{F}_1 \underset{\text{Thm. 5.2.1}}{=} \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_0)) = \bigvee_i \mathcal{M}_{b_i}(\underline{\delta}(\mathcal{F}_0)) = \bigvee_j (x_j = 1)$. After another step, $\underline{\delta}(\mathcal{F}_1) = \mathcal{F}_1$ and so $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_1)) = \mathcal{F}_1$ (see Lemma 2.4.16 below), and the algorithm converges.

As an aside, using Thm. 5.2.1 we can prove the syntactic characterization of the monotone span (Def. 2.4.18), as promised in §2.4:

Proof of Thm. 2.4.19. If $\mathrm{MHull}_B(\varphi) \equiv \varphi$ then by Thm. 5.2.1 φ is equivalent to $\bigwedge\{c \mid c \text{ is a clause, } \varphi \Longrightarrow c, \text{ and } \exists b \in B.\, b \not\models c\}$, which is a CNF representation as desired.

Conversely, if $\varphi \equiv c_1 \wedge \ldots \wedge c_s$ where each clause $c_i \not\models b_i$ for some $b_i \in B$, then in particular $\bigwedge\{c \mid c \text{ is a clause, } \varphi \Longrightarrow c, \text{ and } \exists b \in B.\, b \not\models c\} \subseteq \varphi$, which by Thm. 5.2.1 reads $\mathrm{MHull}_B(\varphi) \subseteq \varphi$, and $\varphi \subseteq \mathrm{MHull}_B(\varphi)$ from Lemma 2.4.14. □

5.3 Abstract Interpretation in The Monotone Span of $\mathcal{B}_k$

In this section, we cast Λ-PDR as an abstract interpretation in a new logical abstract domain of the *monotone span of* $\mathcal{B}_k$. We first discuss abstract interpretation in general, and then develop the notion of a monotone span. We then define the abstract domain and show the connection to Λ-PDR.

5.3.1 Background: Abstract Interpretation

In this section, we review the basics of abstract interpretation [CC77]; see [Urb15, RY20] for complete and general presentations. A *complete join-semilattice* is a tuple $\langle D, \sqsubseteq, \sqcup, \bot \rangle$ where D is partially-ordered by $\sqsubseteq$, $\bigsqcup X$ is the least upper bound of every $X \subseteq D$ ($\forall x \in X.\ x \sqsubseteq \bigsqcup X$ and X is the smallest w.r.t. $\sqsubseteq$ that satisfies this), and $\bot$ is the minimal element ($\forall x \in D.\ \bot \sqsubseteq x$). A *chain* is a sequence of elements from D satisfying $x_1 \sqsubseteq x_2 \sqsubseteq \ldots$, and a *strictly ascending chain* if additionally $x_i \neq x_{i+1}$ for every i. The lattice's *height* is the maximal length of a strictly ascending chain. We consider finite domains, where in particular the height is also finite.

A function $F : D \to D$ is Scott-continuous if for every chain $X = x_0, x_1, \ldots \subseteq D$ it holds $f(\bigsqcup_{x \in X} x) = \bigsqcup_{x \in X} f(x)$. By the Knaster-Tarski theorem, such F has a least fixed-point (lfp)—the least x such that $f(x) = x$—and by Kleene's theorem it is $\bigsqcup_{i \geq 0} F^i(\bot)$. When the domain's height is also finite, the sequence $\{F^i(\bot)\}_{i \geq 0}$ converges to the lfp.

In our setting, the *concrete domain* is the join-semilattice powerset domain of the set of states, $\mathcal{C} = \langle D = 2^{\text{States}[\Sigma]}, \subseteq, \cup, \emptyset \rangle$. An *abstract domain* is a join-semilattice $\mathcal{A} = \langle D^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \bot^\sharp \rangle$. An *abstraction function* is a monotone function $\alpha : D \to D^\sharp$, that is, $\forall S_1 \subseteq S_2 \in D.\ \alpha(S_1) \sqsubseteq^\sharp \alpha(S_2)$. A *conretization function* is a monotone function $\gamma : D^\sharp \to D$, that is, $\forall a_1 \sqsubseteq^\sharp a_2 \in D^\sharp.\ \gamma(a_1) \subseteq \gamma(a_2)$. There is a *Galois connections* between $(D, \subseteq)$ and $(D^\sharp, \sqsubseteq^\sharp)$ through (α, γ), denoted $(D, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (D^\sharp, \sqsubseteq^\sharp)$, if $\forall S \in D, a \in D^\sharp.\ \alpha(S) \sqsubseteq^\sharp a \Leftrightarrow S \subseteq \gamma(a)$.

Let (Init, δ) be a transition system. Define the concrete transformer $F_{\mathit{Init},\delta} : D \to D$ by $F_{\mathit{Init},\delta}(S) = \delta(S) \cup \mathit{Init}$. It is Scott-continuous, since for every increasing $\subseteq$-chain $X \subseteq D$, we have $F_{\mathit{Init},\delta}(\bigcup_{S \in X} S) = \bigcup_{S \in X} F_{\mathit{Init},\delta}(S)$. Its fixed-point $\mathit{lfp}(F_{\mathit{Init},\delta})$ is the set of reachable states of (Init, δ). The corresponding *best abstract transformer* is given by $F^\sharp_{\mathit{Init},\delta}(a) = \alpha(F_{\mathit{Init},\delta}(\gamma(a)))$. $F^\sharp_{\mathit{Init},\delta}$ is also Scott-continuous, and there is a *fixed-point transfer* from F to $F^\sharp$: $\mathit{lfp}(F^\sharp_{\mathit{Init},\delta}) = \alpha(\mathit{lfp}(F_{\mathit{Init},\delta}))$. It follows that $\mathit{lfp}(F^\sharp_{\mathit{Init},\delta})$ is the least abstraction of the set of reachable states, and it is obtained by the chain

$$\bot^\sharp \sqsubseteq^\sharp (F^\sharp_{\mathit{Init},\delta})^1(\bot^\sharp) \sqsubseteq^\sharp (F^\sharp_{\mathit{Init},\delta})^2(\bot^\sharp) \sqsubseteq^\sharp \ldots$$

at its convergence point in a finite i^* with $(F^\sharp_{\mathit{Init},\delta})^{i^*} = (F^\sharp_{\mathit{Init},\delta})^{i^*+1}$ (due to the finite height of $D^\sharp$). We call the chain the *Kleene iterations with the best transformer*, and overall it converges to the most precise sound inductive invariant in the abstract domain.

Another way to phrase the same chain, denoting $\xi_i = (F^\sharp_{Init,\delta})^i(\bot^\sharp)$, is by

$$\xi_1 = \alpha(Init) \qquad \xi_{i+1} = \xi_i \sqcup^\sharp \alpha(\delta(\gamma(\xi_i)) = \alpha(\underline{\delta}(\gamma(\xi_i))),$$

because $\xi_i \sqsubseteq^\sharp \xi_{i+1}$ implies that $\gamma(\xi_i) \subseteq \gamma(\xi_{i+1})$ and therefore $\xi_{i+1} = F^\sharp_{Init,\delta}(\xi_i) = F^\sharp_{Init,\delta}(\xi_i) \sqcup^\sharp \alpha(\delta(\gamma(\xi_i)) \cup Init) = \alpha(\delta(\gamma(\xi_i)) \cup Init \cup \gamma(\xi_i)) = \alpha(\underline{\delta}(\gamma(\xi_i)))$.

5.3.2 Abstract Interpretation in the Monotone Span

For a set of states B, recall that $\mathrm{MSpan}(B)$, the monotone span of B, is the set of formulas φ s.t. $\varphi \equiv \mathrm{MHull}_B(\varphi)$ (Def. 2.4.18), or, equivalently, φ can be represented as a conjunction of clauses, each of which excludes a state from B (Thm. 2.4.19).

We define the abstract domain $\mathbb{M}[B] = \langle \mathrm{MSpan}(B), \Longrightarrow, \sqcup_B, \mathit{false}\rangle$, a logical abstract domain [GMT08] consisting of the monotone span, ordered by logical implication, with bottom element *false*. The existence of $\sqcup_B$ relies on Lemma 2.4.20 (the least upper-bound is the conjunction of all upper-bounds), and $\mathit{false} \in \mathrm{MSpan}(B)$ because $\mathrm{MHull}_B(\mathit{false}) = \mathit{false}$, seeing that $\mathcal{M}_b(\mathit{false}) = \mathit{false}$ for every b.

To define a Galois connection [CC77] between sets of concrete states and formulas in $\mathrm{MSpan}(B)$, we use the *concretization* $\gamma(\varphi) = \{\sigma \mid \sigma \models \varphi\}$ (in the sequel, we refer to γ as the identity function, by our convention of equating formulas with the set of states they represent). The best abstraction is expressed by $\alpha_B(S) = \mathrm{MHull}_B(S)$:

Lemma 5.3.1. *Let $S \subseteq States[\Sigma]$. Then $\mathrm{MHull}_B(S)$ is the least overapproximation of S in $\mathrm{MSpan}(B)$, namely, $\mathrm{MHull}_B(S) \Longrightarrow \varphi$ for every $\varphi \in \mathrm{MSpan}(B)$ s.t. $S \Longrightarrow \varphi$.*

Proof. Since $\varphi \in \mathrm{MSpan}(B)$, by Thm. 5.2.1, $\varphi \equiv \bigwedge_i c_i$ where each clause c_i excludes some $a_i \in B$. If $\alpha_k(S) \not\Longrightarrow \varphi$, then there is some c_i and a corresponding $a_i \in B$ that it excludes such that $\alpha_B(S) \not\Longrightarrow c_i$. This means that $\mathcal{M}_b(S) \not\Longrightarrow c_i$ for every $b \in B$. But then in particular $\mathcal{M}_{a_i}(S) \not\Longrightarrow c_i$, even though $S \Longrightarrow c_i$ and $a_i \not\models c_i$, which is a contradiction to Corollary 2.4.8 for $\mathcal{M}_{a_i}(S)$. □

Lemma 5.3.2. *There is a Galois connection $(2^{States[\Sigma]}, \subseteq) \underset{\alpha_B}{\overset{\gamma}{\leftrightarrows}} (\mathrm{MSpan}(B), \Longrightarrow)$.*[8]

Proof. α_B is monotone by Lemma 2.4.15. γ is also monotone. Let $\varphi \in \mathrm{MSpan}(B)$ and $S \subseteq \mathrm{States}[\Sigma]$. If $\alpha_B(S) \Longrightarrow \varphi$ then $S \subseteq \gamma(\varphi)$ since $S \subseteq \mathrm{MHull}_B(S)$ (Lemma 2.4.14). If $S \subseteq \gamma(\varphi)$ then $\alpha_B(S) \Longrightarrow \varphi$ by Lemma 5.3.1. □

Remark 5.3.3 (Disjunctive completion)**.** *When $B = b_1 \vee \ldots \vee b_m$ is a disjunction of multiple cubes, the domain is not disjunctively-complete [CC79]: if $\varphi_1, \varphi_2 \in \mathrm{MSpan}(B)$, it could be that $\varphi_1 \vee \varphi_2 \notin \mathrm{MSpan}(B)$. However, for a single cube b, the join operation of $\mathbb{M}[b]$ is disjunction, as follows from Lemma 2.4.7. In this case, a definition of the abstraction α_b through the*

[8] Quotient over logical equivalence, this is a Galois *insertion*, as $\alpha_B(\gamma(\psi)) = \mathrm{MHull}_B(\psi) \equiv \psi$ for $\psi \in \mathrm{MSpan}(B)$.

representation function $\beta_b(\sigma) = cube_b(\sigma)$ *is straightforward, since* $\alpha_b(S) = \bigsqcup_{\sigma\in S}\beta_b(\sigma)$ *reads as Corollary 2.4.8.*

Remark 5.3.4 (As a reduced product domain)**.** *One way to understand* $\mathbb{M}[B]$ *when* $B = b_1 \vee \ldots \vee b_m$ *is as a reduced product [CC79] of the per-cube domains:* $\mathbb{M}[B]$ *(quotient on logical equivalence) is isomorphic to* $\bigotimes_i \mathbb{M}[b_i]$*. The Cartesian product domain is over m-tuples of formulas,* $\bigtimes_i \mathrm{MSpan}(b_i)$*, ordered by* $(\varphi_1^1,\ldots,\varphi_m^1) \sqsubseteq (\varphi_1^2,\ldots,\varphi_m^2) \iff \bigwedge_{i=1}^m(\varphi_i^1 \implies \varphi_i^2)$*, with concretization* $\gamma_{\times b_i}(\varphi_1,\ldots,\varphi_m) = \bigcap_{i=1}^m \gamma(\varphi_i)$*. The reduced product quotients the Cartesian product w.r.t. having the same concretization. This is isomorphic to* $\mathbb{M}[B]$ *because* $\gamma_{\times b_i} = \gamma(\bigwedge_{i=1}^m \varphi_i)$*, and if* $\varphi_i \in \mathrm{MSpan}(b_i)$ *then* $\bigwedge_{i=1}^m \varphi_i \in \mathrm{MSpan}(B)$*.*

Λ-PDR as Kleene iterations. Alg. 19 shows Kleene iterations in $\mathbb{M}[\mathcal{B}_k]$ with the best abstract transformer for $(Init, \delta)$. The next iterate (line 6) is always $\mathcal{F}^{\mathrm{ai}}_{i+1} = \alpha_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}^{\mathrm{ai}}_i))$, which exactly matches the relation between successive frames in Λ-PDR (Corollary 5.2.2). This means that Λ-PDR's frames exactly match the Kleene iterates, at least when the initial states are in $\mathrm{MSpan}(\mathcal{B}_k)$ themselves (in which case the first iterate in line 4 is simply *Init*); otherwise there is a difference of at most one frame:

Algorithm 19 Kleene Iterations in $\mathbb{M}[\mathcal{B}_k]$

```
1: procedure Λ-PDR(Init, δ, Bad, k)
2:     i ← 0
3:     F^ai_{-1} ← false
4:     F^ai_0 ← α_{B_k}(Init)
5:     while F^ai_i ⇏ F^ai_{i-1} do
6:         F^ai_{i+1} = α_{B_k}(δ(F^ai_i))
7:         i ← i + 1
8:     return F^ai_i
```

Theorem 5.3.5. $\mathcal{F}_i^{ai} \subseteq \mathcal{F}_{i+1} \subseteq \mathcal{F}_{i+1}^{ai}$ *for every* i *where* $\mathcal{F}_{i+1}$ *exists. Further, if* $Init \in \mathrm{MSpan}(\mathcal{B}_k)$*, then* $\mathcal{F}_i = \mathcal{F}_i^{ai}$*.*

Proof. By induction on i, first prove that $\mathcal{F}_i \subseteq \mathcal{F}^{\mathrm{ai}}_i$ for every i where $\mathcal{F}_i$ exists. Initially, $\mathcal{F}_0 = Init \subseteq \alpha_{\mathcal{B}_k}(Init) = \mathcal{F}^{\mathrm{ai}}_0$. For the step, assume that $\mathcal{F}_i \subseteq \mathcal{F}^{\mathrm{ai}}_i$. Using Lemma 2.4.15, this implies that $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) \subseteq \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}^{\mathrm{ai}}_i))$, which by Corollary 5.2.2 and the Kleene iterations means that $\mathcal{F}_{i+1} \subseteq \mathcal{F}^{\mathrm{ai}}_{i+1}$. For the other inclusion, likewise, since $\mathcal{F}^{\mathrm{ai}}_0 = \alpha_{\mathcal{B}_k}(Init) = \mathrm{MHull}_{\mathcal{B}_k}(Init) \subseteq \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(Init)) = \mathcal{F}_1$ (where the inclusion uses Lemma 2.4.15), we have by induction that $\mathcal{F}^{\mathrm{ai}}_i \subseteq \mathcal{F}_{i+1}$ for every i s.t. $\mathcal{F}_{i+1}$ exists. Similarly, if $Init \in \mathrm{MSpan}(\mathcal{B}_k)$ then $\alpha_{\mathcal{B}_k}(Init) = Init$, and by induction $\mathcal{F}_i = \mathcal{F}^{\mathrm{ai}}_i$ for every i. □

We can now relate the number of frames in Λ-PDR and the number of Kleene iterations:

Corollary 5.3.6. Λ-PDR$(Init, \delta, Bad, k)$ *(Alg. 18) converges or fails (line 8) in a frame whose index is at most one greater than the number of Kleene iterations in* $\mathbb{M}[\mathcal{B}_k]$ *on* $(Init, \delta)$ *(Alg. 19).*

Proof. Let i^* be the iteration where Alg. 19 converges to the least-fixed point, i.e. $\mathcal{F}^{\mathrm{ai}}_{i^*+1} = \mathcal{F}^{\mathrm{ai}}_{i^*}$. If Alg. 18 does not terminate with error (line 8) before $i = i^* + 1$, by Thm. 5.3.5 $\mathcal{F}^{\mathrm{ai}}_i \subseteq \mathcal{F}_{i+1} \subseteq \mathcal{F}^{\mathrm{ai}}_{i+1}$ for every $i \leq i^*$, and for $i = i^*$ we have $\mathcal{F}^{\mathrm{ai}}_{i^*} \subseteq \mathcal{F}_{i+1} \subseteq \mathcal{F}^{\mathrm{ai}}_{i^*+1} = \mathcal{F}^{\mathrm{ai}}_{i^*}$, so $\mathcal{F}_{i^*+1}$ is an inductive invariant. Therefore, $\underline{\delta}(\mathcal{F}_{i^*+1}) = \mathcal{F}_{i^*+1}$, and thus, because $\mathcal{F}_{i^*+1} \in \mathrm{MSpan}(\mathcal{B}_k)$, also $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_{i^*+1})) = \mathcal{F}_{i^*+1}$ and the algorithm converges. □

Further, Λ-PDR converges whenever the Kleene iterations converge to an inductive invariant:

Corollary 5.3.7. *If there exists an inductive invariant $I \in \mathrm{MSpan}(\mathcal{B}_k)$ for $(Init, \delta, Bad)$, then* Λ-PDR$(Init, \delta, Bad, k)$ *(Alg. 18) converges to an inductive invariant.*

Proof. Alg. 19 converges from below to the abstract lfp for $(Init, \delta)$ in $\mathbb{M}[\mathcal{B}_k]$, which from the premise is strong enough to prove safety. Using Thm. 5.3.5, the same is true for Alg. 18. □

Increasing k refines $\mathbb{M}[\mathcal{B}_k]$. Increasing the backward exploration bound k refines $\mathbb{M}[\mathcal{B}_k]$ by increasing $\mathrm{MSpan}(\mathcal{B}_k)$ ($\mathcal{B}_k \subsetneq \mathcal{B}_{k'}$ implies $\mathrm{MSpan}(\mathcal{B}_k) \subsetneq \mathrm{MSpan}(\mathcal{B}_{k'})$: every $\varphi \in \mathrm{MSpan}(\mathcal{B}_k)$ is also $\varphi \in \mathrm{MSpan}(\mathcal{B}_{k'})$, and for instance the clause $\neg\sigma_b \in \mathrm{MSpan}(\mathcal{B}_{k'}) \setminus \mathrm{MSpan}(\mathcal{B}_k)$ if the state $\sigma_b \in \mathcal{B}_{k'} \setminus \mathcal{B}_k$). Restarting Λ-PDR with a larger k (line 9 in Alg. 18) thus refines the domain until it includes an inductive invariant that establishes safety. Such a k always exists because the set of all backward reachable states (attained by some finite k in the setting of propositional systems) is sufficient to express the weakest safe inductive invariant.

Efficient convergence. Unfortunately, the lattice height of $\mathrm{MSpan}(B)$ is exponential. For example, all the formulas in the strictly ascending chain of formulas $\{\overline{x} \leq i\}_{0 \leq i \leq 2^n - 1}$ over n variables $\overline{x} = x_{n-1}, \ldots, x_0$ are in $\mathrm{MSpan}(\{0 \ldots 0\})$ (see §5.8). Therefore, to bound the number of iterations we need to consider properties of the transition system, a task on which we embark next.

5.4 Convergence Bounds via Abstract Diameter

In this section and the next, we prove a bound on the number of iterations of Λ-PDR on a given transition system via the DNF size of a monotonization of the transition relation. In the current section we assume that $\mathcal{B}_k$ can be expressed as a single cube b, and generalize it in §5.5.

To formulate the bound, we define a monotonization of the transition relation, which is a formula over $\Sigma \cup \Sigma'$. For cubes c_1 and c_2 over Σ, we denote by $\mathcal{M}_{(c_1,c_2)}(\delta)$ the monotonization $\mathcal{M}_a(\delta)$ where $a = c_1 \wedge c_2'$ and c_2' is obtained from c_2 by substituting each $p \in \Sigma$ by the corresponding $p' \in \Sigma'$.

The monotonization we perform on the pre-state vocabulary Σ uses the *reflection* of the backward reachable cube:

Definition 5.4.1 (Reflection)**.** *For a cube $b = \ell_1 \wedge \ldots \wedge \ell_r$, the* reflection *is $Ref(b) = \neg\ell_1 \wedge \ldots \wedge \neg\ell_r$.*

Our main theorem in this section is as follows.

Theorem 5.4.2. *Let $(Init, \delta, Bad)$ be a transition system, and $\mathcal{B}_k = b$ a cube. Then* Λ-PDR$(Init, \delta, Bad, k)$ *converges or fails in a frame whose index is bounded by $\left|\mathcal{M}_{(Ref(b),b)}(\delta)\right|_{\mathrm{dnf}} + 1$.*

Example 5.4.3. *For an example where Thm. 5.4.2 yields a tight bound, consider a counter over $\overline{x} = x_n, \ldots, x_0$ where Init is $\overline{x} = 00 \ldots 00$, δ increments even numbers by two, Bad is $\overline{x} = 10 \ldots 01$, and $k = 0$. The monotonization $\delta^{\mathbf{x}} = \mathcal{M}_{\overline{x}=01\ldots10,\overline{x}'=10\ldots01}(\delta)$ is $x_0' = 0$ (see the*

calculation below), so $\left|\mathcal{M}_{\overline{x}=01\ldots10,\overline{x}'=10\ldots01}(\delta)\right|_{\text{dnf}} = 1$. *By Thm. 5.4.2, Λ-PDR converges in* $\mathcal{F}_2$, *and indeed in this example* $\mathcal{F}_1 = (x_n = 0 \wedge x_0 = 0)$, *and* $\mathcal{F}_2 = (x_0 = 0)$ *is the inductive invariant (all the even numbers).*

To see that indeed $\delta^{\#} = (x_0' = 0)$, *note that* $(01\ldots10, 10\ldots0) \in \delta$, *the monotonization* $cube_{\overline{x}=01\ldots10,\overline{x}'=10\ldots00)}(\overline{x} = 01\ldots10, \overline{x}' = 10\ldots01) = (x_0' = 0)$ *and use Corollary 2.4.8; for the other direction,* $x_0' = 0$ *holds in every transition of* δ *and remains in every such monotone cube, invoking again Corollary 2.4.8.*

Example 5.4.4. *An example where Thm. 5.4.2 yields a polynomial yet non-tight bound appears in §5.1.6.*

Outline. We prove Thm. 5.4.2 by constructing an "abstract" transition system whose diameter is the number of iterations required for Alg. 19 to converge (§5.4.1), and bound its diameter (§5.4.2). Throughout, we fix a transition system $(Init, \delta, Bad)$ and a backwards bound $k \in \mathbb{N}$, denoting the cube $\mathcal{B}_k$ by b.

5.4.1 Abstract Transition System

Given $(Init, \delta, Bad)$, the *abstract transition system* $(Init^{\sharp}, \delta^{\sharp}, Bad^{\sharp})$ is defined over the same set of states as the original, and extends its transitions:

Definition 5.4.5 (Abstract Transition System)**.** *The abstract transition system of* $(Init, \delta, Bad)$ *w.r.t. b is defined as a transition system* $(Init^{\#}, \delta^{\#}, Bad^{\#})$ *over* $States[\Sigma]$ *with* $Init^{\#} = \mathcal{M}_b(Init)$, $Bad^{\#} = Bad$, *and* $\delta^{\#} = \mathcal{M}_{(Ref(b),b)}(\delta)$.

The monotonization in $\delta^{\#}$ of the pre-image is understood using the following technical lemma about monotonization w.r.t. a reflection.

Lemma 5.4.6. $\sigma_1 \models cube_{Ref(b)}(\sigma_2) \iff \sigma_2 \models cube_b(\sigma_1)$ *for every cube b and states* σ_1, σ_2.

Proof. Suppose that $\sigma_1 \models cube_{Ref(b)}(\sigma_2)$. Then for every p present in $Ref(b)$ (equivalently, in b), $\sigma_1, Ref(b)$ *dis*agree on p whenever $\sigma_2, Ref(b)$ do. Contrapositively, $\sigma_2, Ref(b)$ *agree* on p whenever $\sigma_1, Ref(b)$ do. Equivalently, σ_2, b *dis*agree on p whenever σ_1, b do. This shows that $\sigma_2 \models cube_b(\sigma_1)$. The other direction of the implication is symmetric because $Ref(Ref(b)) = b$. □

The central property of the abstract transition system is that its i-reachable state capture executions of iterations in the $\mathbb{M}[\mathcal{B}_k]$ abstract domain.

Lemma 5.4.7. *Let* $R_i^{\#}$ *be the set of states reachable in* $(Init^{\#}, \delta^{\#}, Bad^{\#})$ *w.r.t.* $\mathcal{B}_k = b$ *(Def. 5.4.5) in at most i steps. Then* $R_i^{\#} \equiv \mathcal{F}_i^{ai}$ *where* $\mathcal{F}_i^{ai}$ *is the i'th iterate of Alg. 19 in* $\mathbb{M}[b]$ *on* $(Init, \delta, Bad)$.

This will imply that the diameter of the abstract transition system—the minimal i where i-reachability converges to all the reachable states—equals the number of iterations needed for convergence of the abstract interpretation in $\mathbb{M}[\mathcal{B}_k]$.

Before proving this connection, let us explain the intuition for the abstract transition system and the relation to the algorithm.

Through Corollary 2.4.8 and Lemma 5.4.6 we can see that a transition (σ, σ') of $\delta^{\times}$ consists of three steps:

- monotonization of σ to $\widetilde{\sigma}$ (σ is the "protector" state for $\widetilde{\sigma}$); then
- a concrete transition $(\widetilde{\sigma}, \widetilde{\sigma}') \in \delta$; and
- the monotonization of $\widetilde{\sigma}'$ to σ' ($\widetilde{\sigma}'$ is the "protector" state for σ').

The monotonization *after* the concrete transition mimics a step of the algorithm, which computes the post-image and then a monotone overapproximation. A potentially critical difference between the algorithm and $\delta^{\times}$ is that the transition system performs this state-by-state, while the algorithm computes these operations over sets. The insight is that when $\mathcal{B}_k$ is a single cube, the abstraction of Λ-PDR factors to individual states as well, and so can be captured using an ordinary transition system, so that its i-reachable states correspond to iterations of the algorithm, which is the objective of the next lemma.

As the proof shows, the monotonization *before* the concrete transition does not change reachability, because in execution traces of $\delta^{\times}$, this monotonization is absorbed by the monotonization at the end of the previous transition (the first step in the trace is handled by taking the monotonization of the initial states, $\mathit{Init}^{\sharp} = \mathit{Init}^{\times} = \mathcal{M}_b(\mathit{Init})$). Even though the monotonization of the pre-state does not change the diameter, it can improve (and never worsen) our diameter *bound*, which is derived in §5.4.2.

Proof of Lemma 5.4.7. We first prove a similar result for a slightly simpler, "less abstract" transition system, where monotonization is performed in the post-state but not in the pre-state. Define a transition system $(\mathit{Init}^{\sharp}, \delta^{\sharp}, \mathit{Bad}^{\sharp})$ over States$[\Sigma]$ by $\mathit{Init}^{\sharp} = \mathcal{M}_b(\mathit{Init})$, $\mathit{Bad}^{\sharp} = \mathit{Bad}$, and

$$(\sigma, \sigma') \in \delta^{\sharp} \iff \exists \sigma''.\ (\sigma, \sigma'') \in \delta \wedge \sigma' \in \mathit{cube}_b(\sigma'').$$

(In fact, $\delta^{\sharp} = \mathcal{M}_{(\mathit{true},b)}(\delta)$.)

Denote by $R_i^{\sharp}$ the set of states reachable in $(\mathit{Init}^{\sharp}, \delta^{\sharp}, \mathit{Bad}^{\sharp})$ in at most i steps. We argue that $R_i^{\sharp} \equiv \mathcal{F}_i^{\mathrm{ai}}$, by induction on i. Initially, $R_0^{\sharp} = \mathit{Init}^{\sharp} = \mathcal{M}_b(\mathit{Init}) = \mathcal{F}_0^{\mathrm{ai}}$. For the step, by the definition of the abstract system, by the induction hypothesis $R_i^{\sharp} \equiv \mathcal{F}_i^{\mathrm{ai}} \in \mathrm{MSpan}(b)$. Hence,

$$R_{i+1}^{\sharp} = \underline{\delta^{\sharp}}(R_i^{\sharp}) = R_i^{\sharp} \cup \bigvee_{\sigma'' \in \delta(R_i^{\sharp})} \mathit{cube}_b(\sigma'') \underset{\text{Corollary 2.4.8}}{=} R_i^{\sharp} \cup \mathcal{M}_b(\delta(R_i^{\sharp}))$$
$$\underset{R_i^{\sharp} \in \mathrm{MSpan}(b)}{=} \mathcal{M}_b(R_i^{\sharp}) \cup \mathcal{M}_b(\delta(R_i^{\sharp})) \underset{\text{Lemma 2.4.7}}{=} \mathcal{M}_b(\underline{\delta}(R_i^{\sharp})) \underset{\text{ind.}}{=} \mathcal{M}_b(\underline{\delta}(\mathcal{F}_i^{\mathrm{ai}})) = \mathcal{F}_{i+1}^{\mathrm{ai}}.$$

It remains to show that $R_i^{\times} = R_i^{\sharp}$, i.e., that the i-reachable states of $\delta^{\sharp}, \delta^{\times}$ coincide (although they are not in general bisimilar).

First, $\delta^\sharp \subseteq \delta^\times$. This is because if $(\sigma, \sigma') \in \delta^\sharp$, then by definition there is σ'' such that $(\sigma, \sigma'') \in \delta$ and $\sigma' \in \mathit{cube}_b(\sigma'')$. Considering the product monotone order, $(\sigma, \sigma'') \leq_{(\cdot,b)} (\sigma, \sigma')$, and so $(\sigma, \sigma'') \in \delta \Longrightarrow (\sigma, \sigma') \in \mathcal{M}_{(\mathit{Ref}(b),b)}(\delta) = \delta^\times$, as required.

Second, we show that for any $S \in \mathrm{MSpan}(b)$ it holds that $\underline{\delta^\times}(S) \subseteq \underline{\delta^\sharp}(S)$. $\underline{\delta^\times}(S) = S \cup \delta^\times(S)$ and $\underline{\delta^\sharp}(S) = S \cup \delta^\sharp(S)$, so we need to show that $\delta^\times(S) \subseteq \delta^\sharp(S)$. Let $(\sigma, \sigma') \in \delta^\times, \sigma \in S$. By the definition of $\delta^\times$ and Corollary 2.4.8, there exists $(\widetilde{\sigma}, \widetilde{\sigma}') \in \delta$ such that $\sigma \models \mathit{cube}_{\mathit{Ref}(b)}(\widetilde{\sigma})$ and $\sigma' \models \mathit{cube}_b(\widetilde{\sigma}')$. The former implies, by Lemma 5.4.6, that $\widetilde{\sigma} \models \mathit{cube}_b(\sigma)$, hence $\widetilde{\sigma} \in S$ as well (because $S \in \mathrm{MSpan}(b)$). Writing $\sigma'' = \widetilde{\sigma}'$ shows that $(\widetilde{\sigma}, \sigma') \in \delta^\sharp$, and hence $\sigma' \in \underline{\delta^\sharp}(S)$, as required.

The first part of the argument (and induction on i) shows that $R_i^\sharp \subseteq R_i^\times$. We have shown that $R_i^\sharp = \mathcal{F}_i^{\mathrm{ai}}$, which in particular implies that always $R_i^\sharp \in \mathrm{MSpan}(b)$; therefore, the second argument above shows that $R_i^\times \subseteq R_i^\sharp$. The claim follows. □

Corollary 5.4.8. *Let $(\mathit{Init}^\times, \delta^\times, \mathit{Bad}^\times)$ be the abstract transition system w.r.t. $\mathcal{B}_k = b$ (Def. 5.4.5). If $(\mathit{Init}^\times, \delta^\times, \mathit{Bad}^\times)$ is safe and its reachability diameter is s, then Λ-PDR$(\mathit{Init}, \delta, \mathit{Bad}, k)$ converges in frame at most $s+1$. If $(\mathit{Init}^\times, \delta^\times, \mathit{Bad}^\times)$ reaches a bad state in s steps, then Λ-PDR$(\mathit{Init}, \delta, \mathit{Bad}, k)$ fails (line 8) in frame at most $s+1$.*

Proof. From Lemma 5.4.7, $\mathcal{F}_s^{\mathrm{ai}} \equiv \mathcal{F}_{s+1}^{\mathrm{ai}}$ iff $R_s^\times \equiv R_{s+1}^\times$, and the least s in which the latter holds is the diameter. For the unsafe case, $\mathcal{F}_s^{\mathrm{ai}} \cap \mathit{Bad} \neq \emptyset$ iff $R_s^\times \cap \mathit{Bad} \neq \emptyset$. Apply Corollary 5.3.6 in both cases to deduce convergence, resp. failure, of Λ-PDR in frame at most $s+1$. □

5.4.2 Diameter Bound via Abstract DNF Size

In this section, we bound the diameter of the abstract transition system in order to obtain the convergence bound of Thm. 5.4.2. We use a simple, general bound on the diameter of transition systems, by the DNF size of the transition relation:

Lemma 5.4.9. *The reachability diameter of a transition system $(\mathit{Init}, \delta, \mathit{Bad})$ is bounded by $|\delta|_{\mathrm{dnf}}$.*

Proof. Fix a minimal DNF representation of δ. Thinking about each disjunct of δ as an action a, every transition can be labeled by at least one action. Whenever in an execution $\sigma_1, \sigma_2, \ldots$ an action a labels two transitions $\sigma_{i_1} \xrightarrow{a} \sigma_{i_1+1}, \sigma_{i_2} \xrightarrow{a} \sigma_{i_2+1}$, the segment between the occurrences, $\sigma_{i_1+1}, \ldots, \sigma_{i_2}$ can be dropped and the resulting trace is still valid (and terminates at the same state)—this is because if $(\sigma_{i_1}, \sigma_{i_1+1}) \models a$ and likewise $(\sigma_{i_2}, \sigma_{i_2+1}) \models a$ then also $(\sigma_{i_1}, \sigma_{i_2+1}) \models a$, because a, which is a cube, can be decomposed to $a_{\mathit{pre}} \wedge a_{\mathit{post}}$ where all the literals in a_{pre} are in Σ and those in a_{post} are in Σ'. Overall, every state that can be reached from another state can do so by an execution where each action appears at most once, and thus the diameter is bounded by $|\delta|_{\mathrm{dnf}}$. □

Combining the above results yields a proof of this section's main theorem:

Proof of Thm. 5.4.2. By Corollary 5.4.8, the number of iterations before convergence or failure of Λ-PDR is bounded by 1 plus the reachability diameter of $(Init^{\mathbf{x}}, \delta^{\mathbf{x}}, Bad^{\mathbf{x}})$, which by Lemma 5.4.9 is at most $\left|\mathcal{M}_{(Ref(b),b)}(\delta)\right|_{\text{dnf}}$. □

Complexity. Finding whether there is an equivalent DNF representation with at most s terms is complete for the second level of the polynomial hierarchy Σ_2^P [Uma01]. This is on par with deciding whether the diameter is bounded by s [HHTW10] (see also [SU02]). Thus the bound in Thm. 5.4.2 is not an efficiently-computable upper bound on the number of frames of Λ-PDR. Instead, we view the result of Thm. 5.4.2 as a conceptual explanation of how smaller diameters can originate from the abstraction.

Example 5.4.10. *In the running example from §5.1 (without the additional transitions in §5.1.6), Thm. 5.4.2 yields a trivial, exponential, bound which is not tight. Consider the system in Fig. 5.1 restricted to the $\overline{x}$ part (see Remark 5.5.5 and the justification in §5.1.6). Then $\delta^{\mathbf{x}} = \mathcal{M}_{\overline{x}=01\ldots11,\overline{x}'=10\ldots00}(\delta) = (x_n = 1 \wedge x'_n = 0) \vee (x_n = 0 \wedge x'_0 = 1) \vee (x_n = 0 \wedge x'_n = 0 \wedge \overline{x}' > \overline{x}) \vee (x_n = 1 \wedge x'_n = 1 \wedge \overline{x}' > \overline{x})$, which has an exponential DNF size,[9] yielding an exponential bound on the number of frames of Λ-PDR. However, as Example 5.2.3 shows, Λ-PDR converges in this case in a constant number of frames.*

To see that indeed $\delta^{\mathbf{x}}$ is as stated above, it is easiest to think about the behavior of $\delta^{\mathbf{x}}$ as an abstract transition system (Def. 5.4.5).

- *Starting in a state σ with $x_n = 0$, $\delta^{\mathbf{x}}$ can lead us to any state σ' with $x'_n = 0$ and $\overline{x}' > \overline{x}$—to reach $\overline{x}'$ the abstraction step turns all the bits below its leading 1, a concrete step increments the counter, so the resulting state agrees with the leading 1 and everything else is 0, and then another abstraction step can generate the other 1's present in $\overline{x'}$.*

  Additionally, we can reach in the abstraction the state $01\ldots11$, a step then skip and arrives at $10\ldots01$, which is abstracted to all numbers with $x_0 = 1$.

  These are all the states we may reach this way; the first abstract step cannot change x_n, and we cannot arrive at smaller numbers, because both the concrete and abstract steps (which turn 0's to 1's) can only strictly increase the number.

- *Starting in a state σ with $x_n = 1$, $\delta^{\mathbf{x}}$ can lead us to any state with $x'_n = 1, \overline{x}' > \overline{x}$, similarly to above for the case $x_n = 0$. Additionally, we can abstract to $11\ldots11$, from which a concrete step wraparounds to $00\ldots00$, which then abstracts to $x'_n = 0$.*

[9] $\overline{x}' > \overline{x}$ is unate (Def. 2.3.4)—it is closed under turning variables in $\overline{x}$ from 1 to 0, and turning variables in $\overline{x}'$ from 0 to 1. Hence its unique and minimal DNF representation consists of the disjunction of all its prime implicants (Corollary 2.3.5). It suffices to show that there is an exponential number of prime implicants. To see this, let v be an assignment to all the variables except the least significant, let $\sigma = (v, 0), \sigma' = (v', 1)$ (so in σ, $\overline{x} = 2v$, and in σ', $\overline{x} = 2v + 1$). Then $(\sigma, \sigma') \models \overline{x}' > \overline{x}$. The only prime implicant that can be obtained by dropping literals from (σ, σ') is the conjunction that includes every $x_i = 0$ when $v[x_i] = 0$ and $\overline{x}' = 1$ when $v[x_i] = 1$—dropping any one of these variables would result in a term that is satisfied by $(v[x_i \mapsto 1], 0), (v'[x_i \mapsto 0], 1)$, which does not satisfy $\overline{x}' > \overline{x}$, and so the resulting term would not be an implicant. Thus, the prime implicant associated with v can be used to reconstruct v, setting $v[x_i] = 0$ when $x_i = 0$ is present in the prime implicant, and $v[x_i] = 1$ when $x'_i = 1$ is present. There are exponentially many choices of v and we have shown that each induces a different prime implicant, and the claim follows.

These are all the states that we may reach this way; we cannot reach smaller numbers with $x'_n = 1$, along a similar argument to the case above of $x_n = 0$.

5.5 Convergence Bounds via Abstract Hypertransition Systems

In this section, we generalize the results of §5.4 to the case that $\mathcal{B}_k$ is not expressible as a single cube. In this case, our bound is the product of monotonizations w.r.t. the different cubes that comprise $\mathcal{B}_k = b_1 \vee \ldots \vee b_m$ in the post-state, and w.r.t. (the reflection of) the least cube that contains all $\mathcal{B}_k$ in the pre-state, defined as follows:

Definition 5.5.1. *If $\mathcal{B}_k = b_1 \vee \ldots \vee b_m$, we denote by $\boxed{\mathcal{B}_k} = \bigcap_{i=1}^m b_i$ (as sets of literals) the cube that consists of the literals that appear in all $b_1, \ldots, b_m$.*

Fix a representation $\mathcal{B}_k = b_1 \vee \ldots \vee b_m$. Our main theorem is as follows:

Theorem 5.5.2. *Let $(Init, \delta, Bad)$ be a transition system. Then Λ-PDR$(Init, \delta, Bad, k)$ converges or fails in a frame whose index is bounded by*

$$\zeta + 1 \stackrel{\text{def}}{=} \prod_{i=1}^{m} \left(\left| \mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}), b_i)}(\delta) \right|_{\text{dnf}} + |\mathcal{M}_{b_i}(Init)|_{\text{dnf}} \right) + 1.$$

The reasons for $\mathcal{M}_{b_i}(Init)$ and $\boxed{\mathcal{B}_k}$ will become clear in §5.5.1. Often *Init* is a cube, in which case $|\mathcal{M}_{b_i}(Init)|_{\text{dnf}} = 1$.

Example 5.5.3. *For an example where Thm. 5.5.2 yields a polynomial convergence bound, consider a counter over $\overline{x} = x_n, \ldots, x_0$ (similar in spirit to Fig. 5.1) with $Init = \overline{x} = 0$, $Bad = \overline{x} = 10\ldots0$, and δ that (i) skips every multiple of 2^r except $\overline{0} = 0\ldots0$; (ii) from every state with $x_r = x_{r-1} = \ldots = x_1 = x_0 = 1$ may also "bounce back" to a state with the same upper bits ($i > r$) and exactly one lower bit ($i \leq r$) is 1, or to $\overline{x} = 0$ if the upper bits are already 0; and (iii) transitions from any multiple of 2^r but $\overline{0}$ to any other multiple of 2^r (including the bad state). We call the set of states between consecutive multiples of $\overline{x} = 2^r, \overline{x} = 2^{r+1}$ a "segment".*

Assume that $r = n - polylog(n)$ (the counter skips relatively few times). We now compute the bound resulting from the theorem. For every $k \geq 1$, $\mathcal{B}_k = \bigvee_{i=s}^n b_i$ where $b_i = (x_{r-1} = 0 \wedge \ldots \wedge x_0 = 0 \wedge x_i = 1)$ (all multiples of 2^r except $0\ldots0$). $\boxed{\mathcal{B}_k} = (x_{r-1} = 0 \wedge \ldots \wedge x_0 = 0)$ (all multiples of 2^r). We find a DNF representation for $\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}), b_i)}(\delta)$ using Lemma 2.4.7 similarly to §5.1.6: The number of segments is 2^{n-r}, and in each segment the abstraction of the "bounce back" transitions subsume the transitions between numbers in the same segments. This amounts to $O(r2^{n-r})$ terms. The number of disjuncts b_i is $n - r$, the number of terms in $\mathcal{M}_{b_i}(Init)$ is 1 because Init is itself a term, and overall Thm. 5.5.2 yields the bound $O(r(n-r)2^{n-r}) = poly(n)$.

Example 5.5.4. *For an example where the theorem yields an exponential convergence bound, consider the same system as in the previous example (Example 5.5.3) but when $r = polylog(n)$. The above calculation still yields the same bound but now it is $\Omega(2^n)$. This exponential bound*

reflects true exponential behavior of the algorithm, because each post-image crosses to at most one new segment, and the abstraction never produces states in a segment beyond those represented in the current frame, mandating at least 2^{n-r} *frames.*[10]

Remark 5.5.5. *There are cases where it is possible to apply Thm. 5.5.2 on a restriction of* δ *to specific values to some of the variables, and this produces a better bound. Suppose that for a set of variables* $\overline{x}$ *and some* $\overline{v}$ *valuation thereof,* $\overline{x} = \overline{v}$ *is an inductive invariant for the system, and* $\exists b \in \mathcal{B}_k.\ b[\overline{x}] = \mathit{Ref}(\overline{v})$*. (In §5.1.6, actually* $\mathcal{B}_k \implies \overline{x} = \mathit{Ref}(\overline{v})$*.) Then applying Thm. 5.5.2 to* $\delta\big|_{\overline{x}\leftarrow\overline{v}} = \delta[\overline{v}/\overline{x}]$*, eliminating* $\overline{x}$ *by substituting* $\overline{v}$ *for it, also yields an upper bound on the number of iterations of* Λ*-PDR. The benefit is that* $\left|\mathcal{M}_{(\dots)}(\delta\big|_{\overline{x}\leftarrow\overline{v}})\right|_{\mathrm{dnf}}$ *can be smaller than with the original* δ*. It is correct to apply the theorem to the restriction and deduce a bound for the original, because under the above premises, always* $\mathcal{F}_i[\delta] = \mathcal{F}_i[\delta\big|_{\overline{x}\leftarrow\overline{v}}] \wedge \overline{x} = \overline{v}$*, where* $\mathcal{F}_i[\tau]$ *is the* i*th frame of* Λ*-PDR w.r.t. transition relation* τ*.*[11]

Outline. To prove Thm. 5.5.2, the first step is to define an analog to the abstract transition system from Def. 5.4.5 that captures Alg. 19 in the general case. If $\mathcal{B}_k$ has a DNF form with m cubes, this can be done using a hypertransition system of width m (§5.5.1). We then proceed to bound its diameter (§5.5.2).

5.5.1 Abstract Hypertransition System

We consider hypertransition systems that are dual to the classical definition [e.g. LL90], in that the pre-state, instead of the post-state, of a hypertransition consists of a set of states.

Definition 5.5.6 (Hypertransition System)**.** *A* hypertransition system *(of width* $m \in \mathbb{N}$*) over* $\mathit{States}[\Sigma]$ *is a tuple* $(\mathit{Init}, \delta, \mathit{Bad})$ *where*

- $\mathit{Init} \subseteq \mathit{States}[\Sigma]$ *is the set of initial states,*
- $\mathit{Bad} \subseteq \mathit{States}[\Sigma]$ *is the set of bad states, and*
- $\delta \subseteq \mathit{States}[\Sigma]^m \times \mathit{States}[\Sigma]$ *is a hypertransition relation. As a formula, it is defined over* m *copies of* Σ *for the pre-states,* $\Sigma_1, \dots, \Sigma_m$*, and a copy* Σ' *for the post-state.*

An execution *of the system is a tree in which the leaves are states from Init, the relationship between a node* σ' *and its children* $\sigma_1, \dots, \sigma_m$ *is that* $(\sigma_1, \dots, \sigma_m, \sigma') \models \delta$*. A state* σ *is* reachable

[10] To see that $\mathrm{MHull}_{\mathcal{B}_k}(\mathcal{F}_i)$ never introduces states in a segment that was not already present in $\mathcal{F}_i$, note that because always $0\dots0 \in \underline{\delta}(\mathcal{F}_i)$ (the initial state), $\mathcal{M}_{0\dots0}(\mathcal{F}_i) = \mathit{true}$, and thus $\mathrm{MHull}_{\mathcal{B}_k\cup\{0\dots0\}}(\underline{\delta}(\mathcal{F}_i)) = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) \wedge \mathcal{M}_{0\dots0}(\underline{\delta}(\mathcal{F}_i)) = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$. Hence, $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) = \mathrm{MHull}_{\mathcal{B}_k\cup\{0\dots0\}}(\underline{\delta}(\mathcal{F}_i)) = \mathcal{M}_{x_{r-1}=0\wedge\dots\wedge x_0=0}(\underline{\delta}(\mathcal{F}_i))$. But the cube $x_{r-1} = 0 \wedge \dots \wedge x_0 = 0$ does not mention the upper bits, and thus the monotonization does not alter these bits, and includes only segments that were already present in $\underline{\delta}(\mathcal{F}_i)$.

[11] This is because $\mathcal{F}_i \overset{\text{def}}{=} \mathcal{F}_i[\delta] \implies \overline{x} = \overline{v}$ by induction on i—since $\overline{x} = \overline{v}$ is an inductive invariant, this holds initially, as well as $\underline{\delta}(\mathcal{F}_i) \implies \overline{x} = \overline{v}$. Now $\mathcal{F}_{i+1} = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) \implies \overline{x} = \overline{v}$ as well, because from the assumption there is $b \in \mathcal{B}_k$ s.t. $b[\overline{x}] = \mathit{Ref}(\overline{v})$, so $\mathit{cube}_b(\sigma) \implies \overline{x} = \overline{v}$ for every $\sigma \in \underline{\delta}(\mathcal{F}_i)$ because in σ, $\overline{x} = \overline{v}$, which are opposite in b and thus retained. Hence $\mathcal{M}_b(\underline{\delta}(\mathcal{F}_i)) = \bigvee_{\sigma\in\underline{\delta}(\mathcal{F}_i)} \mathit{cube}_b(\sigma) \implies \overline{x} = \overline{v}$, and thus also the conjunction $\mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) \implies \overline{x} = \overline{v}$.

in at most i steps *if there is an execution with root σ and height at most i. A state is* reachable *if it is reachable in at most i steps for some $i \in \mathbb{N}$. The* reachability diameter *of the system is the least i such that every reachable state is reachable in i steps.*

A standard transition system is a hypertransition system with width $m = 1$.

Definition 5.5.7 (Abstract Hypertransition System)**.** *The abstract hypertransition system $(Init^{\mathbf{x}}, \delta^{\mathbf{x}}, Bad^{\mathbf{x}})$ of a (standard) transition system $(Init, \delta, Bad)$ w.r.t. $\mathcal{B}_k = b_1 \vee \ldots \vee b_m$ is defined over $States[\Sigma]$ by $Init^{\mathbf{x}} = \mathrm{MHull}_{\mathcal{B}_k}(Init)$, $Bad^{\mathbf{x}} = Bad$, and*

$$(\sigma_1, \ldots, \sigma_m, \sigma') \in \delta^{\mathbf{x}} \iff (\sigma_1, \sigma') \in \mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}), b_1)}(\delta \vee Init') \wedge \ldots \wedge (\sigma_m, \sigma') \in \mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}), b_m)}(\delta \vee Init').$$

The central property of the abstract hypertransition system is that its i-reachable state capture the Kleene iterations in the $\mathbb{M}[\mathcal{B}_k]$ abstract domain.

Lemma 5.5.8. *Let $R_i^{\mathbf{x}}$ be the set of states reachable in $(Init^{\mathbf{x}}, \delta^{\mathbf{x}}, Bad^{\mathbf{x}})$ w.r.t. $\mathcal{B}_k$ (Def. 5.5.7) in at most i steps. Then $R_i^{\mathbf{x}} \equiv \mathcal{F}_i^{ai}$ where $\mathcal{F}_i^{ai}$ is the i'th iterate of Alg. 19 on $(Init, \delta, Bad)$ in $\mathbb{M}[\mathcal{B}_k]$.*

This will imply that the diameter of the abstract hypertransition system equals the number of iterations needed for convergence of the abstract interpretation in $\mathbb{M}[\mathcal{B}_k]$.

Before proving this connection, let us explain the intuition for the abstract hypertransition system and the relation to the algorithm.

Through Corollary 2.4.8 and Lemma 5.4.6 we can see that a hypertransition $(\sigma_1, \ldots, \sigma_m, \sigma')$ of $\delta^{\mathbf{x}}$ consists of three segments:

- the monotonization w.r.t. $\boxed{\mathcal{B}_k}$ of each σ_i to $\widetilde{\sigma}_i$ (σ_i is the "protector" state for $\widetilde{\sigma}_i$); then
- from each resulting state $\widetilde{\sigma}_i$, either a concrete transition $(\widetilde{\sigma}_i, \widetilde{\sigma}_i') \in \delta$, or going back to an initial state $\widetilde{\sigma}_i' \in Init$; and
- application of the monotone hull to arrive at σ', using $\widetilde{\sigma}_1', \ldots, \widetilde{\sigma}_m'$ as "protector" states, each $\widetilde{\sigma}_i'$ showing that σ' is in the monotone overapproximation w.r.t. one of the cubes b_i composing $\mathcal{B}_k$.

The abstraction in the *last* step connects the reachable states of $\delta^{\mathbf{x}}$ and the Kleene iterations of Alg. 19. The idea is that $\sigma' \in \mathrm{MHull}_{\mathcal{B}_k}(\{\widetilde{\sigma}_1', \ldots, \widetilde{\sigma}_m'\})$, and if $\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_m \in \mathcal{F}_i^{\mathrm{ai}}$ then by Lemma 2.4.13 this implies that $\sigma' \in \mathrm{MHull}_{\mathcal{B}_k}(\delta(\mathcal{F}_i^{\mathrm{ai}}) \cup Init)$, which, using the results of §5.3, is the next iterate $\mathcal{F}_{i+1}^{\mathrm{ai}}$. The key point is that the converse also holds—the monotone hull $\mathrm{MHull}_{\mathcal{B}_k}(\delta(\mathcal{F}_i^{\mathrm{ai}}) \cup Init)$ "factors" to $\mathrm{MHull}_{\mathcal{B}_k}(\{\widetilde{\sigma}_1', \ldots, \widetilde{\sigma}_m'\})$ on all m choices of protectors $\widetilde{\sigma}_1', \ldots, \widetilde{\sigma}_m' \in \delta(\mathcal{F}_i^{\mathrm{ai}}) \cup Init$ (this is reminiscent of Carathéodory's theorem in convex analysis). Unlike in Def. 5.4.5, in this definition the protector states also come directly from *Init*, not only from a transition of δ, and essentially this is the reason for $\mathcal{M}_{b_i}(Init)$ in the bound of Thm. 5.5.2,

unlike in Thm. 5.4.2. This is necessary here to "mix" the different protector states, which do not necessarily all originate from the same frame.

As in Def. 5.4.5, the abstraction in the *first* step, which uses $\boxed{\mathcal{B}_k}$, does not change reachability and the diameter, but it can improve our diameter *bound*, which is derived in §5.5.2. This is achieved by also allowing a hypertransition from $(\sigma_1, \ldots, \sigma_m)$ to σ' if other steps of $\delta^{\mathbf{x}}$ (concrete/init, monotone hull) can arrive at σ' from $(\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_m)$ and if we know for certain that whenever $\sigma_1, \ldots, \sigma_m$ are reachable, then so are $\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_m$. This is the case when $\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_m$ belong to $\mathrm{MHull}_{\mathcal{B}_k}(\sigma_1, \ldots, \sigma_m)$, because, as explained above, a monotone hull is performed in the last step of $\delta^{\mathbf{x}}$. Reachability is not extended, because the additional abstraction in the pre-state could be mimicked by an abstraction in the post-state of the previous (abstract) step. One way to ensure that $\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_m$ belong to $\mathrm{MHull}_{\mathcal{B}_k}(\sigma_1, \ldots, \sigma_m)$ is that each $\widetilde{\sigma}_i \in \mathrm{MHull}_{\mathcal{B}_k}(\{\sigma_i\})$, namely, that for every b_j, $\sigma_i \in \mathcal{M}_{b_j}(\sigma'_i)$. This is achieved when $\sigma_i \in \mathcal{M}_{\boxed{\mathcal{B}_k}}(\sigma'_i)$, i.e., $\sigma'_i \in \mathcal{M}_{Ref(\boxed{\mathcal{B}_k})}(\sigma_i)$, which is the reason for using $\boxed{\mathcal{B}_k}$ in the monotonization of the pre-states. Yet:

Example 5.5.9. *In some cases, the abstraction using $\boxed{\mathcal{B}_k}$ is weak, and is another source of non-tightness in the the bound of Thm. 5.5.2. Consider the system from Example 5.2.4. In this case $\boxed{\mathcal{B}_k} = true$, resulting in no abstraction of the pre-state vocabulary. The DNF size of $\mathcal{M}_{(true,b_i)}(\delta)$ is superpolynomial*[12]*, leading to a superpolynomial bound on the number of frames. However, Λ-PDR with $k = 0$ converges in $\mathcal{F}_1$ (see Example 5.2.4).*

Remark 5.5.10. *At first sight, it would seem that the product of monotonizations in the bound of Thm. 5.5.2 is unnecessary, and that one could study the convergence of Λ-PDR w.r.t. $\mathcal{B}_k$ by the convergence w.r.t. the simpler (and larger) set $\boxed{\mathcal{B}_k}$. Since $\mathcal{B}_k \subseteq \boxed{\mathcal{B}_k}$, the overapproximation is tighter with $\boxed{\mathcal{B}_k}$ (Lemma 2.4.15), so it would seem that the number of iterations with $\mathrm{MHull}_{\mathcal{B}_k}(\cdot)$ must be less than with $\mathrm{MHull}_{\boxed{\mathcal{B}_k}}(\cdot)$. However, this is not so. The reason is that Λ-PDR with $\mathrm{MHull}_{\boxed{\mathcal{B}_k}}(\cdot)$ might be converging to an inductive invariant that is not present in $\mathrm{MSpan}(\mathcal{B}_k)$: $\mathcal{B}_k \subseteq \boxed{\mathcal{B}_k}$ implies $\mathrm{MSpan}(\boxed{\mathcal{B}_k}) \subseteq \mathcal{B}_k$. Thanks to such "new" invariants, convergence could be faster with $\mathrm{MHull}_{\boxed{\mathcal{B}_k}}(\cdot)$ than with $\mathrm{MHull}_{\mathcal{B}_k}(\cdot)$.*

We now formally prove the connection between reachable states of the abstract transition system and iterations of the algorithm.

Proof of Lemma 5.5.8. We first prove a similar result for a slightly simpler, "less abstract" hypertransition system, where abstraction is performed in the post-state but not in the pre-state. Define a hypertransition system $(\mathit{Init}^\sharp, \delta^\sharp, \mathit{Bad}^\sharp)$ over $\mathrm{States}[\Sigma]$ $\mathit{Init}^\sharp = \mathrm{MHull}_{\mathcal{B}_k}(\mathit{Init})$,

[12] Let σ be a state $x_i = 0$, and σ' obtained by applying δ with two $x_{j_1}, x_{j_2} \neq x_i$ variables that are 0 in σ. A DNF representation must have a term d_σ such that $(\sigma, \sigma') \models d$. However, d must include all variables $x_r \notin \{x_i, x_{j_1}, x_{j_2}\}$ that are 0 in σ; otherwise $\tilde{\sigma}$ where they are turned off also yields $(\tilde{\sigma}, \sigma') \models d_\sigma$. But this can't be because the x_r's are also 0 in σ', and $\mathcal{M}_{(true,b_i)}(\delta)$ doesn't allow turning 1 bits to 0 (except for x_i). Consider now two states σ_1, σ_2 with $x_i = 1, x_{j_1} = x_{j_2} = 0$, and each has additional $n/2$ variables $x^s_{r_1}, \ldots, x^s_{r_{n/2}} \notin \{x_i, x_{j_1}, x_{j_2}\}$ of value 0 ($s \in \{1, 2\}$), and the rest of the variables are 1. By the above argument, d_{σ_1} requires that all literals $x^1_{r_p}$ are 0, which implies that $\sigma_2 \not\models d_{\sigma_1}$ if $\{x^2_{r_1}, \ldots, x^2_{r_{n/2}}\} \not\subseteq \{x^1_{r_1}, \ldots, x^1_{r_{n/2}}\}$. Therefore, every choice of $n/2$ variables yields a non-comparable term, and there are $\binom{n}{n/2} = \Theta\left(4^n/\sqrt{n}\right)$ such choices.

$Bad^\sharp = Bad$, and

$$(\sigma_1,\ldots,\sigma_m,\sigma') \in \delta^\sharp \iff \exists \sigma_1'',\ldots,\sigma_m''. \quad ((\sigma_1,\sigma_1'') \in \delta \vee \sigma_1'' \in Init) \wedge \sigma' \in cube_{b_1}(\sigma_1'') \quad \wedge \\ \ldots \\ ((\sigma_m,\sigma_m'') \in \delta \vee \sigma_m'' \in Init) \wedge \sigma' \in cube_{b_m}(\sigma_m'').$$

(In fact, $\delta^\sharp = \bigwedge_{i=1}^m (\mathcal{M}_{(true,b_i)}(\delta \vee Init'))[\Sigma_i,\Sigma'])$.

Denote by $R_i^\sharp$ the set of states reachable in $(Init^\sharp,\delta^\sharp,Bad^\sharp)$ in at most i steps. We argue that $R_i^\sharp \equiv \mathcal{F}_i^{\mathrm{ai}}$, by induction on i. Initially, $R_0^\sharp = Init^\sharp = \mathrm{MHull}_{\mathcal{B}_k}(Init) = \mathcal{F}_0^{\mathrm{ai}}$. For the step, the set $R_{i+1}^\sharp$ is the set of states reachable in at most $i+1$ steps in the hypertransition system, which is $R_{i+1}^\sharp = R_i^\sharp \cup \delta(R_i^\sharp)$ where $\delta^\sharp(R_i^\sharp)$ is the set of states σ' so that there are $\sigma_1,\ldots,\sigma_m \in R_i^\sharp$ such that $(\sigma_1,\ldots,\sigma_m,\sigma') \in \delta^\sharp$. By the definition of $\delta^\sharp$,

$$\delta^\sharp(R_i{}^\sharp) = \bigvee_{\sigma_1'',\ldots,\sigma_m'' \in \delta(R_i^\sharp)\cup Init} (cube_{b_1}(\sigma_1'') \wedge \ldots \wedge cube_{b_m}(\sigma_m''))$$

By distributivity of conjunction over disjunction,

$$= \left(\bigvee_{\sigma_1'' \in \delta(R_i^\sharp)\cup Init} cube_{b_1}(\sigma_1'')\right) \wedge \ldots \wedge \left(\bigvee_{\sigma_m'' \in \delta(R_i^\sharp)\cup Init} cube_{b_m}(\sigma_m'')\right)$$

By Corollary 2.4.8, this is

$$= \mathcal{M}_{b_1}(\delta(R_i^\sharp) \cup Init) \wedge \ldots \wedge \mathcal{M}_{b_m}(\delta(R_i^\sharp) \cup Init)$$

By Lemma 2.4.13, this amounts to

$$= \mathrm{MHull}_{\mathcal{B}_k}(\delta(R_i^\sharp) \cup Init)$$

which by the induction hypothesis is

$$= \mathrm{MHull}_{\mathcal{B}_k}(\delta(\mathcal{F}_i^{\mathrm{ai}}) \cup Init).$$

In the terminology of §5.3.1, as $\mathcal{F}_i^{\mathrm{ai}} = (F^\sharp)^{i+1}(\bot^\sharp)$, we have obtained $\delta^\sharp(R_i^\sharp) = \alpha_{\mathcal{B}_k}(\delta((F_{Init,\delta}^\sharp)^{i+1}(\bot^\sharp)) \cup Init) = (F_{Init,\delta}^\sharp)^{i+2}(\bot^\sharp)$. Because $(F_{Init,\delta}^\sharp)^{i+1}(\bot^\sharp) \sqsubseteq^\sharp (F_{Init,\delta}^\sharp)^{i+2}(\bot^\sharp)$, the result is that $R_i^\sharp \subseteq \delta(R_i^\sharp)$, and $R_{i+1}^\sharp = \delta(R_i^\sharp) \cup R_i^\sharp = \delta(R_i^\sharp) = \mathcal{F}_{i+1}^{\mathrm{ai}}$, as required.

It remains to show that $R_i^\times = R_i^\sharp$, i.e., that the i-reachable states of $\delta^\sharp,\delta^\times$ coincide.

First, for every set of states S it holds that $\underline{\delta^\sharp}(S) \subseteq \underline{\delta^\times}(S)$. This is because if $(\sigma_1,\ldots,\sigma_m,\sigma') \in \delta^\sharp$, then by definition there are $\sigma_1'',\ldots,\sigma_m''$ such that $(\sigma_i,\sigma_i'') \in \delta \vee Init'$ for every i and $\sigma' \in cube_{b_i}(\sigma_i'')$. Considering the product monotone order, $(\sigma_i,\sigma_i'') \leq_{(\cdot,b_i)} (\sigma_i,\sigma')$, and so $(\sigma_i,\sigma_i'') \in$

$\delta \vee \mathit{Init}' \implies (\sigma_i, \sigma') \in \mathcal{M}_{(\mathit{Ref}(\boxed{\mathcal{B}_k}),b_i)}(\delta \vee \mathit{Init}')$. This for every i; by the definition of $\delta^{\text{ⓧ}}$ this implies that $(\sigma_1, \ldots, \sigma_m, \sigma') \in \delta^{\text{ⓧ}}$. This means that $\sigma' \in \underline{\delta^{\text{ⓧ}}}(S)$ since $\sigma_1, \ldots, \sigma_m \in S$.

Second, we show that for any $S \in \mathrm{MSpan}(\mathcal{B}_k)$ it holds that $\underline{\delta^{\text{ⓧ}}}(S) \subseteq \underline{\delta^{\sharp}}(S)$. Let $(\sigma_1, \ldots, \sigma_m, \sigma') \in \delta^{\text{ⓧ}}$, where $\sigma_1, \ldots, \sigma_m \in S$. By the definition of $\delta^{\text{ⓧ}}$, $(\sigma_i, \sigma') \models \mathcal{M}_{(\mathit{Ref}(\boxed{\mathcal{B}_k}),b_i)}(\delta \vee \mathit{Init}')$, and so there exist $(\widetilde{\sigma}_1, \widetilde{\sigma}'_1), \ldots, (\widetilde{\sigma}_m, \widetilde{\sigma}'_m) \in \delta \vee \mathit{Init}'$ such that for every i,

- $\sigma_i \models \mathit{cube}_{\mathit{Ref}(\boxed{\mathcal{B}_k})}(\widetilde{\sigma}_i)$—by Lemma 5.4.6, this means that $\widetilde{\sigma}_i \models \mathit{cube}_{\boxed{\mathcal{B}_k}}(\sigma_i)$. Since $\sigma_i \in S$, by Corollary 2.4.8, $\widetilde{\sigma}_i \in \mathcal{M}_{\boxed{\mathcal{B}_k}}(S)$. It follows that $\widetilde{\sigma}_i \in \mathrm{MHull}_{\mathcal{B}_k}(S)$ (because $\mathcal{B}_k \subseteq \boxed{\mathcal{B}_k}$ implies $\mathrm{MHull}_{\boxed{\mathcal{B}_k}}(S) \subseteq \mathrm{MHull}_{\mathcal{B}_k}(S)$ and $\mathrm{MHull}_{\boxed{\mathcal{B}_k}}(S) \equiv \mathcal{M}_{\boxed{\mathcal{B}_k}}(S)$ by Lemma 2.4.13). From the premise that $S \in \mathrm{MSpan}(\mathcal{B}_k)$, $\mathrm{MHull}_{\mathcal{B}_k}(S) \equiv S$, and we have $\widetilde{\sigma}_i \in S$.
- $\sigma' \models \mathit{cube}_{b_i}(\widetilde{\sigma}'_i)$.

Writing $\sigma''_i = \widetilde{\sigma}'_i$ shows that $(\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_m, \sigma') \in \delta^{\sharp}$, because for every i we have $\widetilde{\sigma}_i \in S$, $(\widetilde{\sigma}_i, \widetilde{\sigma}'_i) \in \delta \vee \mathit{Init}'$, and $\sigma' \models \mathit{cube}_{b_i}(\widetilde{\sigma}'_i)$. This shows that $\sigma' \in \underline{\delta^{\sharp}}(S)$, as required.

The first part of the argument (and induction on i) shows that $R^{\sharp}_i \subseteq R^{\text{ⓧ}}_i$. We have shown that $R^{\sharp}_i = \mathcal{F}^{\mathrm{ai}}_i$, which in particular implies that always $R^{\sharp}_i \in \mathrm{MSpan}(\mathcal{B}_k)$; therefore, the second argument above shows that $R^{\text{ⓧ}}_i \subseteq R^{\sharp}_i$. The claim follows. □

Corollary 5.5.11. *Let $(\mathit{Init}^{\text{ⓧ}}, \delta^{\text{ⓧ}}, \mathit{Bad}^{\text{ⓧ}})$ be the abstract hypertransition system w.r.t. $\mathcal{B}_k$ (Def. 5.5.7). If $(\mathit{Init}^{\text{ⓧ}}, \delta^{\text{ⓧ}}, \mathit{Bad}^{\text{ⓧ}})$ is safe and its reachability diameter is s, then Λ-PDR$(\mathit{Init}, \delta, \mathit{Bad}, k)$ converges in frame at most $s+1$. If $(\mathit{Init}^{\text{ⓧ}}, \delta^{\text{ⓧ}}, \mathit{Bad}^{\text{ⓧ}})$ reaches a bad state in s steps, then Λ-PDR$(\mathit{Init}, \delta, \mathit{Bad}, k)$ fails (line 8) in frame at most $s+1$.*

Proof. Follows from Lemma 5.5.8 similarly to the proof of Corollary 5.4.8 from Lemma 5.4.7. □

5.5.2 Hyperdiameter Bounds via a Joint Abstract Cover

In this section, we bound the diameter of the abstract transition in order to obtain the convergence bound of Thm. 5.5.2. The proof is based on a diameter bound similar to the case of standard transition systems.

Lemma 5.5.12. *The reachability diameter of a hypertransition system $(\mathit{Init}, \delta, \mathit{Bad})$ is bounded by $|\delta|_{\mathrm{dnf}}$.*

Proof. Fix a minimal DNF representation of δ. Thinking about each disjunct of δ as an action, every transition from m children to a parent can be labeled by at least one action. Consider a path from the root to the leaves in an execution tree. With these actions, if an action a labels two (hyper)transitions $\sigma_{i_1} \xrightarrow{a} \sigma_{i_1+1}, \sigma_{i_2} \xrightarrow{a} \sigma_{i_2+1}$, the segment of the tree between the occurrences, $\sigma_{i_1+1}, \ldots, \sigma_{i_2}$ can be dropped, replacing the and the resulting trace is still valid (and terminates at the same state)—this is because if $(\sigma^1_{i_1}, \ldots, \sigma^m_{i_1}, \sigma_{i_1+1}) \models a$ and likewise $(\sigma^1_{i_2}, \ldots, \sigma^m_{i_2}, \sigma_{i_2+1}) \models a$ then also $(\sigma^1_{i_1}, \ldots, \sigma^m_{i_1}, \sigma_{i_2+1}) \models a$, because a, which is a cube, can be decomposed to $a_{\mathit{pre}_1} \wedge \ldots \wedge a_{\mathit{pre}_m} \wedge a_{\mathit{post}}$ where all the literals in a_{pre_i} are in the i'th pre-state

copy Σ_i and those in a_{post} are in Σ'. Overall, every state that can be reached from a set of leaf states can do so by an execution where each action appears at most once on each path of the tree, and thus the diameter is bounded by $|\delta|_{\text{dnf}}$. □

Proof of Thm. 5.5.2. Denote the bound in the theorem by $q+1$. From the distributivity of the conjunction in $\delta^{\mathbf{x}}$, $|\delta^{\mathbf{x}}|_{\text{dnf}} \leq \prod_{i=1}^{m} \left|\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(\delta \vee Init')\right|_{\text{dnf}}$, and

$$\begin{aligned}\left|\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(\delta \vee Init')\right|_{\text{dnf}} &\underset{\text{Lemma 2.4.7}}{=} \left|\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(\delta) \vee \mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(Init')\right|_{\text{dnf}} \\ &\leq \left|\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(\delta)\right|_{\text{dnf}} + \left|\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(Init')\right|_{\text{dnf}} \\ &= \left|\mathcal{M}_{(Ref(\boxed{\mathcal{B}_k}),b_i)}(\delta)\right|_{\text{dnf}} + |\mathcal{M}_{b_i}(Init)|_{\text{dnf}},\end{aligned}$$

overall yielding that $|\delta^{\mathbf{x}}|_{\text{dnf}} \leq q$.

By Corollary 5.5.11, the number of iterations before convergence or failure of Λ-PDR is bounded by 1 plus the reachability diameter of $(Init^{\mathbf{x}}, \delta^{\mathbf{x}}, Bad^{\mathbf{x}})$, which by Lemma 5.5.12 is at most q.

□

5.6 Complexity Bounds for Λ-PDR

Each iterate in Alg. 19 involves a monotone hull (lines 4 and 6), which is a conjunction of monotonizations. Using our monotonization algorithm, Alg. 15 from Chapter 4, this can be computed efficiently. We follow on this idea to prove efficient complexity upper bounds on Alg. 19, combining the efficiency of this algorithm with the bounds on the number of iterations from §5.5 to prove a complexity upper bound on Λ-PDR.

Continuing to use the same conventions for the DNF representations of $\mathcal{B}_k = b_1 \vee \ldots \vee b_m$ and the abbreviation ζ for the bound on number of iterations. We prove that it is possible to implement the abstract interpretation form of Λ-PDR, Alg. 19, so that its overall complexity is polynomial in the same bound on the number of iterations ζ, the number of variables n, and the number m of terms in the representation of $\mathcal{B}_k$:

Theorem 5.6.1. *Alg. 19 can be implemented to terminate in $O(n^2\zeta + (n+m)\zeta^2)$ SAT queries and time.*

Example 5.6.2. *Continuing the example of Example 5.5.3, where $\zeta = O(r(n-r)2^{n-r}) = poly(n)$, Thm. 5.6.1 shows that Λ-PDR terminates in $O(n^2 r(n-r)2^{n-r} + (n+2^{n-r})r^2(n-r)^2 2^{3(n-r)}) = O(n^2 r^2 (n-r)^2 2^{2(n-r)}) = poly(n)$ SAT calls and time.*

Example 5.6.3. *Let n be an odd number. Consider a transition system over $\overline{x} = x_1, \ldots, x_n$, where Init is $\overline{x} = 00\ldots00$, Bad is $\overline{x} = 11\ldots11$ and the transition relation chooses an even number of variables that are 0 from the initial state and turns them into 1. If we take $k = 0$, then $\mathcal{B}_k$ is a cube ($m = 1$), and Lemma 2.4.7 yields that $\mathcal{M}_{\overline{x}=00\ldots00 \wedge \overline{x}'=11\ldots11}(\delta) = \bigvee_{i=1}^{n}(x_i' = 0)$*

(originating from the transitions from $00\ldots00$ *to everything* 1 *except a single bit, which subsumes transitions from* $00\ldots00$ *to states with fewer* 1*'s) so* $\zeta = |\mathcal{M}_{\overline{x}=00\ldots00\wedge\overline{x}'=11\ldots11}(\delta)|_{\mathrm{dnf}} + 1 = O(n)$*. Thm. 5.6.1 shows that Λ-PDR in this case terminates in* $O(n^3)$ *SAT queries and time. This is significant because a naive implementation of Alg. 19 would start, for the first iteration of line 6, by computing the exact post-image* $\delta(\mathit{Init})$*; in our example this is the set of states where the parity of* $\overline{x}$ *is* 0*, which cannot be represented in polynomial-size DNF nor CNF [e.g. CH11]. Our implementation is able to compute the abstraction of the post-image without constructing the post-image and avoids the blowup in complexity.*

At this point, the direct approach to implement Alg. 19 is to perform $\mathrm{MHull}_B(\varphi)$ in lines 4 and 6 through $\bigwedge_{j=1}^{m}$ Monotonize(φ, b_j), invoking Alg. 15 on φ. Indeed, this achieves a bound that is only slightly worse than Thm. 5.6.1 (see Remark 5.6.6). In what follows we provide an implementation that both explicates the connection to ζ, and achieves exactly the bound of Thm. 5.6.1.

Algorithm 20 Efficient Kleene Iterations in $\mathbb{M}[\mathcal{B}_k]$

1: **procedure** Efficient-Λ-PDR(*Init*, δ, *Bad*)
2: $\quad i \leftarrow 0$
3: $\quad \mathcal{F}^{\mathrm{ai}}_{-1} \leftarrow \mathit{false}$
4: $\quad \mathcal{F}^{\mathrm{ai}}_{0} \leftarrow \bigwedge_{j=1}^{m}$ Monotonize(Init, b_j)
5: $\quad$ **for** $j = 1..m$ **do**
6: $\quad\quad \delta^{\boxtimes}_j \leftarrow$ Monotonize$(\delta \vee \mathit{Init}', \mathit{Ref}(\boxed{\mathcal{B}_k}) \wedge b'_j)$
7: $\quad$ **while** $\mathcal{F}^{\mathrm{ai}}_i \not\Rightarrow \mathcal{F}^{\mathrm{ai}}_{i-1}$ **do**
8: $\quad\quad \mathcal{F}^{\mathrm{ai}}_{i+1} = \bigvee \{ \left(t_1\big|_{\Sigma'}\right) \wedge \ldots \wedge \left(t_m\big|_{\Sigma'}\right) \;\big|\; t_j \text{ a term of } \delta^{\boxtimes}_j,\ \exists \sigma_j \in \xi_i.\, \sigma_j \models \left(t_j\big|_{\Sigma}\right) \}$
9: $\quad\quad i \leftarrow i + 1$
10: $\quad$ **return** $\mathcal{F}^{\mathrm{ai}}_i$

Our implementation is displayed in Alg. 20. The first iterate is computed as described above by invoking Alg. 15 on *Init* (line 4). The SAT queries performed by Alg. 15 are in this case straightforward, with $\varphi = \mathit{Init}$.

To compute the next iterates, we first compute monotnizations of the concrete transformer, $\delta \vee \mathit{Init}'$ (line 6). This is a two-vocabulary formula, and accordingly the monotonizations are w.r.t. two-vocabulary cubes. The monotonizations are computed in DNF form and stored in $\delta^{\boxtimes}_j$ (see §4.6.1 of Chapter 4). The next iterate $\mathcal{F}^{\mathrm{ai}}_{i+1}$ is formed from the $\delta^{\boxtimes}_j$'s by taking all the combinations of terms from $\delta^{\boxtimes}_1, \ldots, \delta^{\boxtimes}_m$ whose pre-state part is satisfied by at least one state in ξ_i, and forming the conjunction of the post-state parts: for a term $t = \ell_1 \vee \ldots \vee \ell_{i_1} \vee \ell'_{i_1+1} \vee \ldots \vee \ell'_{i_2}$ over $\Sigma \uplus \Sigma'$, the restriction $t\big|_{\Sigma} = \ell_1 \vee \ldots \vee \ell_{i_1}$ and $t\big|_{\Sigma'} = \ell'_{i_1+1} \vee \ldots \vee \ell'_{i_2}$.

The invocation of Alg. 15 in line 6 is on a double-vocabulary formula; still, the SAT queries to be performed in the invocation of Alg. 15 are simple SAT queries about two-vocabulary formulas (and a counterexample is a pair of states).

It is important for the efficiency result that Alg. 17 uses Alg. 15 as a subprocedure. Using Bshouty's procedure (see Remark 4.6.6) would yield a bound in terms of the DNF size of the

original transition relation, which could be significantly larger, especially in cases where the abstract interpretation procedure terminates faster than exact forward reachability (as in the running example of Fig. 5.1 or Example 4.2.3).

The rest of this section proves that Alg. 20 realizes Thm. 5.6.1. The correctness of the algorithm is shown in the following lemma:

Lemma 5.6.4. *$\mathcal{F}_i^{ai}$ in Alg. 20 is logically equivalent to $\mathcal{F}_i^{ai}$ in Alg. 19.*

Proof. By induction over i. The correctness of $\mathcal{F}_0^{\text{ai}}$ follows from the correctness of Alg. 15 (Thm. 4.6.1). For the same reasons, $\delta_j^{\boldsymbol{\times}}$ of Alg. 20 is equivalent to $\mathcal{M}_{Ref(\boxed{\mathcal{B}_k})\wedge b'_j}(\delta \vee Init')$. Now for some DNF manipulation: for every σ',

$$
\begin{aligned}
& \exists \sigma_1, \ldots, \sigma_m . (\sigma_1, \sigma') \models \delta_1^{\boldsymbol{\times}} \wedge \ldots \wedge (\sigma_m, \sigma') \models \delta_m^{\boldsymbol{\times}} \\
\iff\ & \exists \sigma_1, \ldots, \sigma_m .\ \exists t_1 \text{ term of } \delta_1^{\boldsymbol{\times}}, \ldots, \exists t_m \text{ term of } \delta_m^{\boldsymbol{\times}} .\ (\sigma_1, \sigma') \models t_1 \wedge \ldots \wedge (\sigma_m, \sigma') \models t_m \\
\iff\ & \exists \sigma_1, \ldots, \sigma_m .\ \exists t_1 \text{ term of } \delta_1^{\boldsymbol{\times}}, \ldots, \exists t_m \text{ term of } \delta_m^{\boldsymbol{\times}} . \\
& \qquad \sigma_1 \models \left(t_1|_{\Sigma}\right) \wedge \sigma' \models \left(t_1|_{\Sigma'}\right) \wedge \ldots \wedge \sigma_1 \models \left(t_m|_{\Sigma}\right) \wedge \sigma' \models \left(t_m|_{\Sigma'}\right) \\
\iff\ & \exists \sigma_1,\ \exists t_1 \text{ term of } \delta_1^{\boldsymbol{\times}} .\ \sigma_1 \models \left(t_m|_{\Sigma}\right) \wedge \sigma' \models \left(t_m|_{\Sigma'}\right) \\
& \wedge \ldots \wedge \\
& \exists \sigma_m,\ \exists t_m \text{ term of } \delta_m^{\boldsymbol{\times}} .\ \sigma_m \models \left(t_m|_{\Sigma}\right) \wedge \sigma' \models \left(t_m|_{\Sigma'}\right) .
\end{aligned}
$$

Hence, $\sigma' \in \mathcal{F}_{i+1}^{\text{ai}}$ of Alg. 20 iff $\exists \sigma_1, \ldots, \sigma_m \in \mathcal{F}_i^{\text{ai}}$ of Alg. 20 that with σ' satisfy

$$
(\sigma_1, \sigma') \models \mathcal{M}_{Ref(\boxed{\mathcal{B}_k})\wedge b'_1}(\delta \vee Init') \wedge \ldots \wedge (\sigma_m, \sigma') \models \mathcal{M}_{Ref(\boxed{\mathcal{B}_k})\wedge b'_m}(\delta \vee Init').
$$

Notice that this corresponds exactly to σ' being in the post-image $\delta^{\boldsymbol{\times}}(\mathcal{F}_i^{\text{ai}})$, with the abstract hypertransition system from §5.5.1. Lemma 5.5.8 shows that this captures exactly $\mathcal{F}_{i+1}^{\text{ai}}$, provided that $\mathcal{F}_i^{\text{ai}}$ is correct, which it is thanks to the induction hypothesis. The claim follows. □

We can now proceed to prove the complexity bound for Alg. 20.

Lemma 5.6.5. *Alg. 20 terminates in $O(n^2\zeta + (n+m)\zeta^2)$ SAT queries and time.*

Proof. By Thm. 4.6.1, each invocation of Alg. 15 in line 4 takes $O\left(n^2 \cdot |\mathcal{M}_{b_i}(Init)|_{\text{dnf}}\right) = O(n^2\zeta)$ queries and time. Similarly, each invocation in line 6 takes $O\left((2n)^2 \cdot \left|\mathcal{M}_{Ref(\boxed{\mathcal{B}_k})\wedge b'_i}(\delta \vee Init')\right|_{\text{dnf}}\right)$ queries and time. This quantity is $O(n^2\zeta)$ through the same calculation as in the proof of Lemma 5.5.8. In each iteration, the number of combinations of terms in line 8 is at most $\prod_{i=1}^{m} \left|\mathcal{M}_{Ref(\boxed{\mathcal{B}_k})\wedge b'_i}(\delta \vee Init')\right|_{\text{dnf}}$. For each of the m terms in the combination, we split the term to Σ, Σ' parts in time linear term size which is at most n, and perform a SAT check for whether the term intersects $\mathcal{F}_i^{\text{ai}}$. Overall this step involves $O\left(m \cdot \zeta\right)$ queries and $O\left((n+m) \cdot \zeta\right)$ time. This is the cost of each iteration; the number of iterations is bounded by ζ by Thm. 5.5.2. The claim follows. □

The proof of Thm. 5.6.1 follows from Lemmas 5.6.4 and 5.6.5.

Remark 5.6.6. *Lemmas 5.6.4 and 5.6.5 have the consequence that in Alg. 19, $|\mathcal{F}_i^{ai}|_{\text{dnf}} \leq \zeta$ (interestingly, this is true in particular for the resulting inductive invariant). This is a proof that the direct implementation of the monotone hull by m calls to Alg. 15 amounts to $O(n^2 m\zeta)$ SAT queries in each iteration, and $O(n^2 m\zeta^2)$ time thanks to Thm. 5.5.2. Though asymptotically inferior, this implementation approach may be more efficient than Alg. 20 when $|\mathcal{F}_i^{ai}|_{\text{dnf}} \ll \zeta$.*

5.7 Forward Reachability in Λ-PDR and Others

This section highlights the importance of the successive overapproximation embodied in the Kleene iterations of Λ-PDR by contrasting Λ-PDR with the treatment of forward reachability in other invariant inference algorithms.

Exact forward reachability. Exact forward reachability iterates $R_0 = \mathit{Init}, R_{i+1} = \underline{\delta}(R_i)$, so that R_i is the set of states reachable in at most i steps (without any overapproximation). We have shown that in some cases Λ-PDR can converge in a significantly lower number of iterations than exact forward reachability, stated formally in the following lemma.

Lemma 5.7.1. *There exists a family of transition systems $(\mathit{Init}, \delta, \mathit{Bad})$ over Σ with $|\Sigma| = n$ and $k = O(1)$ such that Λ-PDR$(\mathit{Init}, \delta, \mathit{Bad}, k)$ converges in poly(n, k) iterations, whereas exact forward reachability converges in $\Omega(2^n)$ iterations.*

Proof. See e.g. §5.1.5 and Example 5.2.3. □

This gap reflects a gap between the diameter of the original system (Init, δ) and the diameter of the abstract system $(\mathit{Init}^{\sharp}, \delta^{\sharp})$ (Def. 5.4.5 and Corollary 5.4.8).

Dual interpolation. The essence of *interpolation-based inference* (ITP) [McM03] is generalizing from proofs of *bounded* unreachability. We consider the time-dual (§2.2.1) of this approach, generalizing from bounded unreachabilty *from* the initial states, rather than unreachability *to* the bad states, in line with our focus here on the treatment of forward reachability.

In §4.3.3 of Chapter 4 we analyzed the time-dual of a model-based ITP algorithm [CIM12, BGKL13] whose generalization procedure was inspired by PDR. The code appears in Alg. 16. Briefly, the algorithm iteratively samples pre-states σ of counterexamples to induction, and excludes each counterexample—similarly to PDR—by seeking a minimal clause c over the literals that are falsified in σ that does not exclude a state from $\mathcal{R}_s$, the set of states that the system can reach in s steps, and conjoins c to the candidate.

We showed there (Thm. 4.6.7) that the algorithm successfully finds an inductive invariant when there an invariant I whose inner-boundary $\partial^+(I)$ (Def. 4.2.1) is s-reachable, that is, $\partial^+(I) \subseteq \mathcal{R}_s$ (this is the forwards fence condition of Def. 4.3.8). In the example of Fig. 5.1 (from §5.1), this does not hold for the invariant in Equation (5.1) unless $s = \Omega(2^n)$ (for example, $\overline{x} = 110\ldots0, \overline{y} = 0\ldots0, z = 0 \in \partial^+(I)$ but reachable only in $\Omega(2^n)$ steps).

In contrast, we prove that for Λ-PDR, it is enough that $\partial^+(I)$ is s-reachable in the *abstract* (hyper)system, which interleaves concrete steps and abstraction (see Def. 5.5.7), and thus can reach $\partial^+(I)$ in fewer steps, which would result in convergence of Λ-PDR with a smaller number of frames:

Theorem 5.7.2. *Let $I \in \mathrm{MSpan}(\mathcal{B}_k)$ be an inductive invariant for $(Init, \delta, Bad)$, and $\mathcal{R}^{\mathsf{x}}_s$ the set of states reachable in at most s steps in $(Init^{\mathsf{x}}, \delta^{\mathsf{x}}, Bad^{\mathsf{x}})$ (Def. 5.5.7). If $\partial^+(I) \subseteq \mathcal{R}^{\mathsf{x}}_s$, then $s+1$ frames suffice for $\Lambda\text{-PDR}(Init, \delta, Bad, k)$ to successfully find an inductive invariant.*

Proof. The proof uses the same property of the monotone hull of the boundary of Lemma 4.6.8 that was also used to prove the result for the model-based dual-ITP algorithm in §4.6.2.

The set of states reachable in s steps in $(Init^{\mathsf{x}}, \delta^{\mathsf{x}}, Bad^{\mathsf{x}})$ is $\mathcal{F}^{\mathrm{ai}}_s$ of the Kleene iterations (Lemma 5.5.8). We can apply Lemma 4.6.8, because $\underline{\delta}(\mathcal{F}_{s-1})$ of Λ-PDR includes all s-reachable states (by properties 1–4 in §5.1.1) and $I \cap \mathcal{B}_k = \emptyset$ since I is an inductive invariant. We obtain that $\mathcal{F}^{\mathrm{ai}}_s = \mathrm{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}^{\mathrm{ai}}_{s-1}))$ contains I. It cannot "overshoot" beyond I due to Corollary 5.3.7. Apply Corollary 5.3.6 for the connection to Λ-PDR. □

In essence, these different criteria for when the forward exploration of the algorithm is sufficient reflect the difference in how the algorithms generalize: per counterexample, both find a minimal clause that does not exclude states from some form of forward reachability, but in Λ-PDR this is an abstraction of forward reachability, whereas model-based dual ITP uses exact forward reachability.

This difference also manifests in different outcomes of Alg. 18 and Alg. 16 on the running example of Fig. 5.1. For every $s < 2^n$ there is an execution of Alg. 16 that fails (line 7) because it includes reachable states as counterexamples to exclude (for example, the first counterexample in the execution of Alg. 16 is $\sigma_b = (\overline{x} = 10\ldots00, \overline{y} = 11\ldots10, z = 1)$, which can be generalized to $c = (x_n = 0)$ that inadvertently excludes also reachable states such as $\overline{x} = 10\ldots01, \overline{y} = 00\ldots00, z = 0)$, although $s = O(1)$ suffices for Λ-PDR (Example 5.2.3).

Finally, we remark that Alg. 16 does use a form of successive overapproximation. By repeatedly generating counterexamples to induction (line 6 of Alg. 16), it in a sense uses reverse frames that overapproximate backward reachability. While both Alg. 16 and Alg. 18 learn lemmas by minimizing a term w.r.t. a forward-reachability analysis in order to block a counterexample from a backward reachability analysis, Alg. 18 employs successive overapproximation is in the former analysis, and Alg. 16 in the latter. As we have seen, this successive overapproximation in counterexample generation is not sufficient for Alg. 16 to successfully infer an invariant for the example of Fig. 5.1. However, it does alleviate the requirement that $I \in \mathrm{MSpan}(\mathcal{B}_k)$, which is necessary in Thm. 5.7.2 but not in Thm. 4.6.7.[13]

[13]The original, non time-dual version of the algorithm is Alg. 13. Its candidates are iteratively increased by forward exploration, so the sequence of candidates is similar to the frames in PDR. However, the roles of backward- and forward-reachability in generalization are reversed. Alg. 13 "overshoots" on the example of Fig. 5.1 unless $s = \Omega(2^n)$, but we focus here on overapproximations that are too tight (rather than too loose), the direction in which Λ-PDR is informative of PDR.

5.8 Between Λ-PDR and PDR: Best Abstraction and Even Better

In each frame, Λ-PDR includes all possible generalizations, which we have shown to amount in $\mathcal{F}_{i+1}$ to the the best abstraction of $\underline{\delta}(\mathcal{F}_i)$ in the abstract domain $\mathbb{M}[\mathcal{B}_k]$ (Lemma 5.3.1). Its frames are thus the strongest (contain fewest states) that satisfy all the properties of frames listed in §5.1.1—the standard ones as well as the monotone span of backward reachable states:

Lemma 5.8.1. *The frames $\mathcal{F}_0, \mathcal{F}_1, \ldots$ of Λ-PDR are the least (w.r.t. $\Longrightarrow$) s.t. for every i, 1. $Init \Longrightarrow \mathcal{F}_0$, 2. $\mathcal{F}_i \Longrightarrow \mathcal{F}_{i+1}$, 3. $\delta(\mathcal{F}_i) \Longrightarrow \mathcal{F}_{i+1}$, and 5. $\mathcal{F}_i \in \mathrm{MSpan}(\mathcal{B}_k)$.*

Proof. That the frames of Λ-PDR satisfy the properties is immediate from the relationship $\mathcal{F}_0 = Init$, $\mathcal{F}_{i+1} = \alpha_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i))$. Minimality is from best abstraction (Lemma 5.3.1) and induction on $i = 0, 1, \ldots$: let $\widetilde{\mathcal{F}}_0, \widetilde{\mathcal{F}}_1, \ldots$ another sequence that satisfies the properties. By property 1, $Init = \mathcal{F}_0 \Longrightarrow \widetilde{\mathcal{F}}_0$. For the step, assume that $\mathcal{F}_i \Longrightarrow \widetilde{\mathcal{F}}_i$. Then from properties 2 and 3, $\underline{\delta}(\widetilde{\mathcal{F}}_i) \Longrightarrow \widetilde{\mathcal{F}}_{i+1}$, and in particular also $\underline{\delta}(\mathcal{F}_i) \Longrightarrow \widetilde{\mathcal{F}}_{i+1}$. From property 5, $\widetilde{\mathcal{F}}_{i+1} \in \mathrm{MSpan}(\mathcal{B}_k)$. Putting these together, Lemma 5.3.1 implies that $\mathcal{F}_{i+1} = \alpha_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) \Longrightarrow \widetilde{\mathcal{F}}_{i+1}$, as required. □

In contrast to Λ-PDR, standard PDR "samples" counterexamples and generalizations, and it does not produce in $\mathcal{F}^{\mathrm{pdr}}_{i+1}$ the least abstraction of $\underline{\delta}(\mathcal{F}^{\mathrm{pdr}}_i)$. Its frames are nevertheless characterized as abstractions (not necessarily the least abstraction) in the same domain:

Lemma 5.8.2. *At any point during the execution of PDR$(Init, \delta, Bad)$ (Alg. 1) when it has at most N frames, $\mathcal{F}^{pdr}_i \in \mathrm{MSpan}(\mathcal{B}_N)$ for every $1 \le i \le N$.*

Proof. In a call to BLOCK$(\sigma_b, i+1)$, it holds $\sigma_b \in \mathcal{B}_{N-i}$, by induction on the recursive calls: The first call BLOCK$(\sigma_b, N+1)$ in line 8 has $\sigma_b \in \mathcal{B}_0 = Bad$, by line 7. In each recursive call from $(\sigma_b, i+1)$ to $(\sigma, i+1-1)$, in line 15 the new counterexample σ reaches the counterexample in the parent call σ_b in one step, so $\sigma_b \in \mathcal{B}_{N-i}$ implies $\sigma \in \mathcal{B}_{N-i+1} = \mathcal{B}_{N-(i-1)}$ as required. Since $\mathcal{B}_{N-i} \subseteq \mathcal{B}_N$, this ensures that $\sigma_b \in \mathcal{B}_N$ in every call to BLOCK$(\sigma_b, i+1)$.

Hence, when the algorithm strengthens $\mathcal{F}^{\mathrm{pdr}}_i$ in line 20, it is always with a clause c such that $\sigma_b \not\models c$ where $\sigma_b \in \mathcal{B}_N$. This implies that $\mathcal{F}^{\mathrm{pdr}}_i \in \mathrm{MSpan}(\mathcal{B}_N)$ (see §2.4.3), completing the proof. □

In particular, this shows that PDR overapproximates the frames that Λ-PDR generates:

Corollary 5.8.3. *At any point during the execution of PDR$(Init, \delta, Bad)$ (Alg. 1) when it has at most N frames, its i'th frame, $\mathcal{F}^{pdr}_i$, satisfies $\mathcal{F}_i \Longrightarrow \mathcal{F}^{pdr}_i$, where $\mathcal{F}_i$ is the i'th frame of Λ-PDR$(Init, \delta, Bad, N)$ (Alg. 18).*

Proof. The frames of Alg. 1 satisfy the properties in the premise of Lemma 5.8.1—all are standard except for the one shown in Lemma 5.8.2.

A more direct argument, outlined in §5.1.3, is that every lemma that PDR learns is also a lemma that Λ-PDR includes in its frames. Formally, to strengthen $\mathcal{F}^{\mathrm{pdr}}_i$, c must block some

$\sigma_b \in \mathcal{B}_n$ and after c is conjoined to the previous frame $\mathcal{F}_{i-1}^{\text{pdr}}$ we must have $\underline{\delta}(\mathcal{F}_{i-1}^{\text{pdr}}) \implies c$; the latter implies, using an induction hypothesis that $\mathcal{F}_{i-1} \implies \mathcal{F}_{i-1}^{\text{pdr}}$, that also $\underline{\delta}(\mathcal{F}_{i-1}) \implies c$. Λ-PDR conjoins *all* such clauses; thus whenever c is conjoined to $\mathcal{F}_i^{\text{pdr}}$, it is also conjoined to $\mathcal{F}_i$. □

In other words, PDR's frame also constitute some sort of search in the abstract domain $\mathbb{M}[\mathcal{B}_N]$ (though in a complex manner, refining previous frame etc.), and its frames always generate at least as much overapproximation as Λ-PDR. Hence, our results that show significant overapproximation in Λ-PDR translate to PDR as well.

Still, the difference between the algorithms is significant—PDR's frames don't employ the best abstraction in this domain. How does this benefit PDR? We show two ways. First, computing all generalizations may be inefficient. Second, it may not be desirable—it could lead to too precise abstraction and slow convergence.

Inefficient frame size. Consider a system over n variables $x_1, \ldots, x_n$, with $\mathit{Init} = x_1 = \ldots = x_n = 0$, $\mathit{Bad} = x_1 = \ldots = x_n = 1$, and δ that non-deterministically chooses some $i \neq j$ with $x_i = x_j = 0$ and sets $x_i \leftarrow 1$.

We start with the analysis of Λ-PDR. In this example, $\mathcal{B}_k = 1 \ldots 1$ (for every k). We argue that $\mathcal{F}_i$ is exactly the set R_i of states reachable in at most i steps, which is the set of states with at most i bits 1, denoted $\{\overline{x} \mid \#1(\overline{x}) \leq i\}$. This can be seen by induction: initially, this holds for $\mathcal{F}_0 = \mathit{Init} = \{0 \ldots 0\}$. In each step $\delta(\mathcal{F}_i) = R_i \cup \{\overline{x} \mid \#1(\overline{x}) = i+1\}$. Then $\mathcal{F}_{i+1} = \text{MHull}_{\mathcal{B}_k}(\underline{\delta}(\mathcal{F}_i)) = \mathcal{M}_{1\ldots1}(\underline{\delta}(\mathcal{F}_i)) = R_i \cup \mathcal{M}_{1\ldots1}(\{\overline{x} \mid \#1(\overline{x}) = i+1\}) = R_{i+1}$, because $\mathcal{M}_{1\ldots1}(\{\overline{x} \mid \#1(\overline{x}) = i+1\})$ adds states that are obtained from a state with $\#1(\overline{x}) = i+1$ by flipping 1's to 0's, resulting in states with smaller values of $\#1(\overline{x})$ that are already included in R_i.

Unfortunately, the set $R_{\lfloor n/2 \rfloor + 1}$ is not expressible in polynomial-size CNF nor DNF.[14] This means that some of Λ-PDR's frames need an exponential number of clauses, and so construct an exponential number of generalizations of the bad state. Even an alternative DNF computation (based on Lemma 2.4.7) would not fare better.

In contrast, $\mathcal{F}_i^{\text{pdr}}$ consists of a single clause blocking the bad state, which is short.

Slow convergence. Consider a counter over $\overline{x} = x_n, x_{n-1}, \ldots, x_0$ with $\mathit{Init} = (\overline{x} = 0 \ldots 0)$, $\mathit{Bad} = (\overline{x} = 1 \ldots 1)$, and δ that increments the counter except for when $\overline{x} = 1 \ldots 10$ which skips the bad state and wraps-around to $0 \ldots 0$.

We start with the analysis of Λ-PDR. Similar to the previous example, $\mathcal{B}_k = 1 \ldots 1$ (for every k) and $\mathcal{F}_i = \mathcal{R}_i$, except that $\mathcal{R}_i$ is now the set of states $\overline{x} \leq i$, because $\delta(\mathcal{F}_i)$ always adds the state $\overline{x} = i+1$, and its $1 \ldots 1$-monotonization adds only states with smaller values of $\overline{x}$ which are already included in $\mathcal{R}_i$ (the derivation is similar to the previous example).

Therefore, the frames $\mathcal{F}_i = \{\overline{x} : \overline{x} \leq i\}$ do not converge until $i = 2^n - 1$, which means that Λ-PDR converges after an exponential number of frames.

[14] It is the majority function, which is not in AC^0 [Hås86], a complexity class that includes poly-size CNF and DNF.

In contrast, in this example, PDR always converges in a *linear* number of frames. The proof uses the fact from Lemma 5.8.2 that the frames of PDR are $(1 \dots 1)$-monotone, and that $\mathcal{F}_i^{\text{pdr}}$ is always exactly one clause, because it blocks the single backward reachable state using a single lemma. Since $\mathcal{F}_i^{\text{pdr}} \implies \mathcal{F}_{i+1}^{\text{pdr}}$, and $\mathcal{F}_i^{\text{pdr}}$ is 1...1-monotone, the clause that is $\mathcal{F}_{i+1}^{\text{pdr}}$ must be a syntactic subset of the clause that is $\mathcal{F}_i^{\text{pdr}}$ [Qui54]. Until they converge, the difference between two successive frames must be that some literals are omitted from the clause, which can happen at most n times.

5.9 Related Work for Chapter 5

PDR as abstract interpretation. This work is not the first to study the relation between PDR and abstract interpretation. Rinetzky and Shoham [RS16] prove that the reachable configurations of PDR are in simulation with the reachable states of a non-standard backward trace semantics. Their work studies standard PDR as non-standard abstract interpretation, whereas we study non-standard PDR as standard abstract interpretation (in a new domain); our domain abstracts the simpler collecting states semantics with standard forward iterations. Our work emphasizes the overapproxmation inherent in the abstraction, where, in particular, the abstraction forces overapproximation in the sequence of frames, whereas Rinetzky and Shoham's property-guided Cartesian trace semantics domain is precise enough to express any sequence of frames that satisfy properties 1–4 from §5.1.1. In contrast, adding property 5 characterizes Λ-PDR as Kleene iterations in our $\mathbb{M}[\mathcal{B}_k]$ domain.

Abstract transition systems. Dams et al. [DGG97] construct, from a transition system and a Galois connection, abstract transition systems that preserve safety and other temporal properties. These are defined over a state space of abstract elements (e.g., formulas in the case of a logical domain), forming abstract edges between abstract elements through $\exists\exists$ or $\forall\exists$ relations of original transitions between the concretizations. It is important for our diameter bounds from the DNF representation of the abstract (hyper)transition system that it is defined over the original state space (Defs. 5.4.5 and 5.5.7), which is possible due to the special structure of $\mathbb{M}[B]$ (see Lemmas 5.4.7 and 5.5.8). In that respect our abstract transition systems are closer to monotonic abstraction in well-structured transition systems by Abdulla et al. [ADHR09], the abstract transition systems for universally-quantified uninterpreted domains by Padon et al. [PIS+16], and the surjective abstraction games of Fecher et al. [FH07].

Diameter bounds. Diameter bounds have been studied in the context of completeness thresholds for bounded model checking [BCCZ99, KS03]. The recurrence diameter [BCC+99, KS03], the longest loop-free path, was studied as a more easily-computable upper bound on the diameter. In our setting, this measure cannot be reduced by the abstraction, which only adds transitions. There are also works that encode the completeness threshold assumption as another verification condition [see DKW08, §IV.D]. Another line of work computes diameter bounds by a composition of diameter bounds of subsystems formed by separating dependencies between

variables in the system's actions [ANG18, BKA02, RG13]. Existing works have considered guarded-update actions, in which variables are either modified to a constant value or remain unchanged; this is not directly applicable to the actions that arise in our abstract transition systems, where monotonization in effect "havocs" variables. Havocked variables are different because in a transition, they *can* change, but not *necessarily*; weaker notions of dependence to capture this may be interesting in future work. We are not aware of a previous application of the $|\delta|_{\text{dnf}}$ diameter bound (Lemma 5.4.9). This bound is never worsened by monotonization, as $|\mathcal{M}_{\dots}(\delta)|_{\text{dnf}} \leq |\delta|_{\text{dnf}}$ [Bsh95, and a corollary of Lemma 2.4.7], and can be exponentially smaller, as e.g. in §5.1.6. The diameter bounds by Konnov et al. [KLVW17, KVW14] share with our work the motivation of analyzing the diameter of abstractions of the original system. They rely on the special structure of counter abstractions of fault-tolerant distributed systems to apply movers [Lip75] and acceleration.

Complexity of PDR. The work by Seufert and Scholl [SS17] includes a complexity analysis of all executions of PDR on the case of two synchronized n-bit counters, where PDR requires an exponential number of SAT calls (this also follows from the fact that the only CNF invariant is exponentially-large) but an enhanced time-dual version of it converges in one frame. Our earlier results in the thesis for maximal transition systems with monotone invariants (Corollary 3.5.8) also involved convergence in essentially one frame. In §5.8 we go beyond this with an analysis of standard PDR on a simple example where convergence requires multiple frames. Our analysis of Λ-PDR centered on the number of frames, not the complexity of constructing them, which is an interesting direction for future work. Although, in the spirit of §5.4, we can bound $|\mathcal{F}_i|_{\text{dnf}} \leq |\delta^{\mathsf{x}}|_{\text{dnf}} = \left|\mathcal{M}_{(Ref(b),b)}(\delta)\right|_{\text{dnf}}$ (when $b = \mathcal{B}_k$ is a cube), the original Λ-algorithm's complexity analysis [Bsh95] for computing $\mathcal{M}_b(\varphi)$ depends on $|\varphi|_{\text{dnf}}$, not $|\mathcal{M}_b(\varphi)|_{\text{dnf}}$, which in our setting is the difference between the concrete and the significantly reduced abstract diameter.

Complexity of abstract interpretation. The efficiency of the abstract transformers is crucial to the overall success of abstract interpretation, which is often at odds with the domain accuracy; a famous example is the octagon abstract domain [Min06], whose motivation is the prohibitive cost of the expressive polyhedra domain [CH78]. We provide a way to compute abstract transformers in the monotone span domain that is efficient in terms of the DNF size of the result (see also Remark 5.6.6). The computation of the abstract transformer in Alg. 20 is inspired by works in symbolic abstraction [RSY04, TLLR15] about finding *representations* of the best abstract transformer, rather than computing it anew per input [ELS+11, TER12, RT16].

The monotone theory in invariant inference. Previous applications of the monotone theory in invariant inference are discussed in the related work for Chapter 4 (§4.8)

Chapter 6

Conclusion

To paraphrase Valiant [Val84], learning theory became possible once precise models became available for modeling the commonplace phenomenon of learning—good models which were interesting to study in their own sake, that promised to be relevant to learning in practice, and that were able to shed light also on the limits of what can be learned. This thesis was a first attempt to achieve a similar feat for the (somewhat less commonplace) phenomenon of invariant inference.

Motivated by the rise of SAT-based invariant inference algorithms, we have attempted to elucidate some of the principles on which they are based by a theoretical complexity analysis of algorithms attempting to infer invariants of polynomial size. In Chapter 3 we developed learning models for invariant inference. Using these we have demonstrated the inherent hardness of the problem and that it is harder than classical learning with queries, which in particular highlights the ambiguity of counterexamples to induction, which has not been formally proven established before. The analysis of different inference models also led us to a surprising result about the power of rich queries for the success of invariant inference algorithms, and the importance of learning from valid (unsat) queries.

At the heart of our analysis in Chapter 3 lies the observation that many interesting SAT-based algorithms can be cast in a black-box model. One avenue for further research is an information-based analysis of black-box models extended with white-box capabilities, e.g. by investigating syntactical conditions on the transition relation that simplify generalization. This is also an interesting question in practice: what additional whitebox analyses can be beneficially integrated with the largely black-box approaches of state-of-the-art inference algorithms?

In Chapter 4 we applied ideas from exact concept learning to understand the behavior of interpolation-based invariant inference, achieving the first polynomial complexity result with access to a SAT oracle for such algorithms, and we extended them to algorithms that can infer in a provably-efficient manner increasingly expressive syntactic classes of invariants. The ability to transfer algorithmic ideas between exact concept learning and invariant inference, and recent results in inferring quantified invariants and invariants over interesting theories [e.g. KBI+17, GSM16, GNMR16, FWSS19, KPIA20, DDLM13, SA16], 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. AAP+13, Ari04]. Conversely, developing exact learning for richer logics can reap benefits in understanding the complexity of invariant inference algorithms in such domains.

The focus of Chapter 4 in on *worst case* complexity. 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 4.3.4). To achieve theoretical guarantees for programs that do not satisfy the fence condition, it may be useful to go beyond worst case—for example, analyzing convergence with high probability. This is an exciting direction for future work, in part due to the potential connections with techniques in learning theory that are based on the analysis of Boolean functions [e.g. O'D14].

Chapter 5 targets property-directed reachability, and distills a previously unknown principle of this algorithmic approach. Through Λ-PDR and its analysis based on the monotone theory from exact learning, we have shown that PDR overapproximates an abstract interpretation process in a new logical abstract domain. We have further shown how this abstraction achieves a significantly more effective forward reachability exploration than approaches that use exact post-image computations or bounded unrollings, and how this can partially be explained through the difference between diameter bounds between the original system and its abstraction. To the author, rationalizing property-directed reachability was a long-held aspiration (if not a personal vendetta) since working on an implementation of the algorithm in [FFL+17].

In future work, it will be interesting to understand the mechanisms by which PDR deviates from naive backward reachability, avoiding the pitfall in the other direction, of overapproximating too much. We hope that this will eventually lead to efficient complexity results for PDR itself. It will also be interesting to study variants of PDR that target infinite-state using richer logics beyond propositional logic. Our observation that there is inherent abstraction in PDR due to states it *cannot* exclude from a frame may also be relevant in such settings. Here too this could have interesting implications back in exact concept learning, requiring extensions of the monotone theory to other logics, which to our knowledge have not been attempted.

We are hopeful that the new view of PDR we provide will inspire practical variations of PDR, improving lemma generation or frame maintenance; that rethinking the principles of PDR can lead to transcending the state-of-the-art. This remains to be seen.

There are several high-level potential takeaways that can be learned from the results in this thesis.

- **Learning invariants**: Learning theory can be an invaluable tool, not only to create new invariant inference algorithms, but also to shed light on existing, longstanding ones (§§4.5 and 5.2).

- **Space & time**: Invariant inference is typically harder than classical learning, because it

needs not only to accommodate complex syntactic forms of invariants, but also navigate the program's behavior over time and complex reachability patterns (§3.6).

- **Which invariants**: It is possible to analyze invariant inference algorithms in a way that shows that they consistently converge to an inductive invariant which is not necessarily the least- or greatest-fixed point (§4.3.2 and Thm. 5.7.2).
- **Syntactic structure**: Existing and new invariant inference algorithms are guaranteed to efficiently infer an inductive invariant of complex syntactic structure, using techniques and ideas inspired by exact concept learning (§§4.3.3 and 4.6).
- **Overapproximation**: Repeated generalization is used by invariant inference to obtain effective overapproximation, and that can be distilled—at least in part—to abstract interpretation that is inextricable from the original approach (§5.3).
- **SAT queries**: Richer SAT queries can enable more powerful algorithms that explore the state space in more sophisticated ways (§3.5).

More broadly, this thesis demonstrates that there is still much to understand about the invariant inference problem, and invariant inference algorithms, using a theoretical toolkit. We believe that additional research in this spirit would not only further illuminate existing verification techniques, but can also give rise to entirely new verification algorithms. By leveraging more advanced classical learning and innovative computational complexity results, the verification algorithms of the future may be transformative to the verification practice by converging to desirable invariants of richer syntactic forms and with more intricate reachability patterns, and at same time have characteristic successes and limitations that are well-understood, thereby guiding verification practitioners when they attempt to prove the correctness of the most challenging hardware and software systems. Such research may also better explain the limitations of fully automatic verification, in a way that could be used to make the most of automation in partly-automatic verification.

Finally, we would like to describe a few well-defined technical questions that arise from this thesis but remain open.

Open question: Lower bounds for polynomial-length transition relations. In our definitions in Chapter 3, the complexity of a black-box query-based inference algorithm (Def. 3.3.5) is a function of the target invariant length derived from the vocabulary size, and does not depend on the length of the representation of the transition relation (which the inference algorithm cannot access directly).

Can inference algorithms utilize an assumption that the transition relation can be expressed by formulas of polynomial size? Formally, this asks for an analysis of the query complexity as it depends also on $|\delta|$ in addition to $|\Sigma|$. It turns out that in this case the algorithm can "breach" the black-box definition, (concept-) learn the transition relation formula itself, and then compute an inductive invariant without additional queries.

Lemma 6.0.1. *There exists a computationally unrestricted Hoare-query inference algorithm $\mathcal{A}^{\mathcal{H}}$ with query complexity polynomial in $|\Sigma|, |TS|$ for the class of transition systems $\mathcal{P}_{\Sigma_2^P}$ (§3.4.1).*

Proof. Using halving/majority vote [BF72, Ang87b, Lit87] and equivalence queries to the transition relation; see the appendix of [FISS20]. □

However, we conjecture that this is inherently possible only due to the use of exponentially-long queries. This is related to the open question in exact concept learning of whether a membership and polynomially-long equivalence queries can efficiently identify general DNF formulas.

Conjecture 6.0.2. *Every Hoare-query inference algorithm $\mathcal{A}^{\mathcal{H}}$,* even computationally-unrestricted, querying on formulas polynomial in $|\Sigma| + |TS|$, *for the class of transition systems $\mathcal{P}_{\Sigma_2^P}$ (§3.4.1) and and for any class of target invariants $\mathcal{L}$ s.t.* Mon-CNF$_n \subseteq \mathcal{L}$, *has query complexity superpolynomial in $|\Sigma| + |TS|$.*

If Conjecture 6.0.2 is true, a result analogous to Thm. 3.4.1 can be obtained, obtaining superpolynomial lower bounds not only in $|\Sigma|$ but also in $|TS|$.

(The original Thm. 3.4.1 is not trivial even without this, because although the set class of all transition systems is doubly-exponential, the class of polynomially-long inductive invariants is only exponentially large, not doubly-exponential. The exponential lower bound we obtain in Thm. 3.5.3 is already exponential also in $|TS|$—and this holds even when candidates can be exponentially long—as the transition relations in $\mathcal{M}_E$ are all of size polynomial in their vocabulary.)

Open question: Bounded model checking and Hoare queries. The algorithms we study in Chapter 4 fall into the black-box model of extended Hoare queries (§3.4.2). Can efficiency results similar to e.g. Thm. 4.3.5 can be obtained in the (non-extended) Hoare-query model—without unrollings for BMC? We believe that the answer is negative. In particular, we conjecture that the extended Hoare-query model is strictly stronger than the standard Hoare-query model:

Conjecture 6.0.3. *There exists a class of transition systems for which*

- *polynomial-length invariant inference has* polynomial *query complexity in the* extended *Hoare-query model, but*
- *every algorithm in the (*standard*) Hoare-query model requires an* exponential *number of queries.*

A precursor is the conjecture that bounded model checking cannot be implemented in general with checks of reachability in a single step:

Conjecture 6.0.4. *There exists a class of transition systems for which, given k and a transition system $(Init, \delta, Bad)$ in the class, it is impossible to answer whether $\underline{\delta}^k(Init) \cap Bad \stackrel{?}{=} \emptyset$ with number of Hoare-queries that is polynomial in $n = |\Sigma|$ and k.*

This is also interesting because PDR is sometimes advocated as a bug-finding procedure, and it uses only standard Hoare-queries.

Open question: Impossibility results under the fence condition. In Chapter 4 we introduced the fence condition as a means to overcome the impossibility results and enable efficient inference of invariants of certain forms. Still, we had to take great care in our inference algorithms, and our translations from exact concept learning to invariant inference, to use only queries of specific one-sided forms, while we required a stronger condition to allow the implementation in invariant inference of *every* exact concept learning algorithm. It seems that this is necessary, but we have not proved it:

Conjecture 6.0.5. *There exists a class of transition systems and a class of polynomial-length invariants such that every safe transition system is known to have an inductive invariant from the class that satisfies the fence condition, and yet every invariant inference algorithm requires a super-exponential number of queries to solve this class.*

Open question: Diameter bounds for abstract transition systems. In Chapter 5 we derived bounds on the number of frames in Λ-PDR by bounding the diameter of a the "abstract" transition system (§5.4). We bounded the diameter of $(Init, \delta)$ by $|\delta|_{\text{dnf}}$; however, this is sometimes sensitive to small changes in the transition relation that do not change the diameter of either the original or abstract transition system (see Examples 5.4.4 and 5.4.10). We conjecture that it is possible to find a diameter bound that favors abstraction (even though in general the diameter of the abstract system might be larger than that of the original), and extensive w.r.t. exact reachability, meaning that it is not sensitive to changes that do not alter exact reachability

Conjecture 6.0.6. *There is a "reasonable" function bound that, given a transition system $(Init, \delta)$, returns a valid bound on the diameter of $(Init, \delta)$, which is*

- *favors abstraction:* $bound(Init^{\mathbf{x}}, \delta^{\mathbf{x}}) \leq bound(Init, \delta)$*, and*
- *extensive:* $bound(Init_1, \delta_1) = bound(Init_2, \delta_2)$ *if* $\forall i.\ \underline{\delta_1}^{Init_1}(i) = \underline{\delta_2}^{Init_2}(i)$.

Bibliography

[AAP+13] Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. 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*, pages 49–60, 2013. 150

[ABD+15] 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. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, pages 1–25. 2015. 3, 7, 43, 70

[ACMN05] Rajeev Alur, Pavol Cerný, P. Madhusudan, and Wonhong Nam. Synthesis of interface specifications for java classes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 98–109, 2005. 70

[ADHR09] Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda, and Ahmed Rezine. Monotonic abstraction: on efficient verification of parameterized systems. *Int. J. Found. Comput. Sci.*, 20(5):779–801, 2009. 147

[AHH13] Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. All for the price of few. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 476–495, 2013. 24

[AHH15] Parosh Abdulla, Frédéric Haziza, and Lukáš Holík. Parameterized verification through view abstraction. *International Journal on Software Tools for Technology Transfer*, pages 1–22, 2015. 24

[ALGC12] Aws Albarghouthi, Yi Li, Arie Gurfinkel, and Marsha Chechik. Ufo: A framework for abstraction- and interpolation-based software verification. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International*

Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings, volume 7358 of *Lecture Notes in Computer Science*, pages 672–678. Springer, 2012. 3

[AM13] Aws Albarghouthi and Kenneth L. McMillan. Beautiful interpolants. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 313–329, 2013. 3, 108

[AMN05] Rajeev Alur, P. Madhusudan, and Wonhong Nam. Symbolic compositional verification by learning assumptions. In *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, pages 548–562, 2005. 70

[Ang87a] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. 70

[Ang87b] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987. xiii, 4, 10, 13, 25, 26, 27, 43, 66, 67, 68, 78, 90, 98, 152

[Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990. 4, 43, 69

[Ang04] Dana Angluin. Queries revisited. *Theor. Comput. Sci.*, 313(2):175–194, 2004. 4

[ANG18] Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Formally verified algorithms for upper-bounding state space diameters. *J. Autom. Reason.*, 61(1-4):485–520, 2018. 148

[AP95] Howard Aizenstein and Leonard Pitt. On the learnability of disjunctive normal form formulas. *Mach. Learn.*, 19(3):183–208, 1995. xiii, 4, 13, 26, 27, 78, 86

[Ari04] Marta Arias. *Exact learning of first-order horn expressions from queries*. PhD thesis, Tufts University, Medford, MA, 2004. 150

[BBL+17] Cristina Borralleras, Marc Brockschmidt, Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 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*, pages 99–117, 2017. 79, 109

[BCC+99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using SAT procedures instead of bdds. In Mary Jane Irwin, editor, *Proceedings of the 36th Conference on Design Automation, New Orleans, LA, USA, June 21-25, 1999*, pages 317–320. ACM Press, 1999. 147

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 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*, pages 193–207, 1999. 11, 20, 147

[BCG02] José L. Balcázar, Jorge Castro, and David Guijarro. A new abstract combinatorial dimension for exact learning via queries. *J. Comput. Syst. Sci.*, 64(1):2–21, 2002. 4

[BDVY17] Nader H. Bshouty, Dana Drachsler-Cohen, Martin T. Vechev, and Eran Yahav. Learning disjunctions of predicates. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 346–369, 2017. 43, 70

[BF72] JM Barzdins and R Freivald. On the prediction of general recursive functions. In *Soviet Mathematics Doklady*, volume 13, pages 1224–1228, 1972. 152

[BG15] Nikolaj Bjørner and Arie Gurfinkel. Property directed polyhedral abstraction. In *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, pages 263–281, 2015. 3

[BGKL13] Nikolaj Bjørner, Arie Gurfinkel, Konstantin Korovin, and Ori Lahav. 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*, pages 35–41, 2013. xiii, 3, 12, 13, 14, 23, 73, 75, 99, 108, 143

[BKA02] Jason Baumgartner, Andreas Kuehlmann, and Jacob A. Abraham. Property checking via structural analysis. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2002. 148

[Bra11] Aaron R. Bradley. Sat-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, pages 70–87, 2011. xiii, 2, 3, 7, 9, 22, 40, 41, 58, 75, 108, 113, 114, 123

[Bra12] Aaron R. Bradley. Understanding IC3. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2012. 9

[Bsh95] Nader H. Bshouty. Exact learning boolean function via the monotone theory. *Inf. Comput.*, 123(1):146–153, 1995. xiii, xiv, 4, 14, 15, 16, 17, 28, 29, 31, 32, 33, 34, 35, 78, 86, 87, 89, 92, 94, 96, 98, 101, 114, 116, 123, 148

[Bsh96] Nader H. Bshouty. A subexponential exact learning algorithm for DNF using equivalence queries. *Inf. Process. Lett.*, 59(1):37–39, 1996. 4

[Bsh97] Nader H. Bshouty. Simple learning algorithms using divide and conquer. *Comput. Complex.*, 6(2):174–194, 1997. 4, 95

[Bsh18] Nader H. Bshouty. Exact learning from an honest teacher that answers membership queries. *Theor. Comput. Sci.*, 733:4–43, 2018. 109

[CBKR19] John Cyphert, Jason Breck, Zachary Kincaid, and Thomas W. Reps. Refinement of path expressions for static analysis. *Proc. ACM Program. Lang.*, 3(POPL):45:1–45:29, 2019. 79, 109

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252, 1977. 3, 16, 24, 126, 127

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Symp. on Princ. of Prog. Lang.*, pages 269–282, New York, NY, 1979. ACM Press. 127, 128

[CCF+10] Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Ming-Hsien Tsai, Yih-Kuen Tsay, and Bow-Yaw Wang. Automated assume-guarantee reasoning through implicit learning. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 511–526, 2010. 108

[CGMT14] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, pages 46–61, 2014. 3

[CGS10] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient generation of craig interpolants in satisfiability modulo theories. *ACM Trans. Comput. Log.*, 12(1):7:1–7:54, 2010. 3, 108

[CH78] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski, editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96. ACM Press, 1978. 24, 148

[CH11] Yves Crama and Peter L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2011. 74, 141

[CIM12] Hana Chockler, Alexander Ivrii, and Arie Matsliah. 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*, pages 72–85, 2012. xiii, 12, 13, 14, 23, 73, 75, 99, 108, 143

[CM78] Ashok K. Chandra and George Markowsky. On the number of prime implicants. *Discret. Math.*, 24(1):7–11, 1978. 86

[Cra57] William Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *J. Symbolic Logic*, 22(3):250–268, 09 1957. 73

[CSS03] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, pages 420–432, 2003. 7

[CTS08] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. Dysy: dynamic symbolic execution for invariant inference. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 281–290, 2008. 69

[DA16] Samuel Drews and Aws Albarghouthi. Effectively propositional interpolants. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*, pages 210–229. Springer, 2016. 3, 12, 108

[DDLM13] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 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*, pages 443–456, 2013. 3, 149

[DGG97] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997. 147

[DKW08] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 27(7):1165–1178, 2008. 147

[DSY17] Dana Drachsler-Cohen, Sharon Shoham, and Eran Yahav. Synthesis with abstract examples. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 254–278, 2017. 70

[ECGN01] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Trans. Software Eng.*, 27(2):99–123, 2001. 69

[ELS+11] Matt Elder, Junghee Lim, Tushar Sharma, Tycho Andersen, and Thomas W. Reps. Abstract domains of affine relations. In Eran Yahav, editor, *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings*, volume 6887 of *Lecture Notes in Computer Science*, pages 198–215. Springer, 2011. 148

[EMB11] Niklas Eén, Alan Mishchenko, and Robert K. Brayton. 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*, pages 125–134, 2011. xiii, 2, 3, 21, 22, 40, 47, 54, 58, 75, 108, 113, 123

[END+18] P. Ezudheen, Daniel Neider, Deepak D'Souza, Pranav Garg, and P. Madhusudan. Horn-ice learning for synthesizing invariants and contracts. *PACMPL*, 2(OOPSLA):131:1–131:25, 2018. 2, 23, 69

[FB18] Grigory Fedyukovich and Rastislav Bodík. 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*, pages 251–269, 2018. 3

[FEM+18] Yotam M. Y. Feldman, Constantin Enea, Adam Morrison, Noam Rinetzky, and Sharon Shoham. Order out of chaos: Proving linearizability using local views. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 23:1–23:21, 2018. 6

[FFL+17] Asya Frumkin, Yotam M. Y. Feldman, Ondrej Lhoták, Oded Padon, Mooly Sagiv, and Sharon Shoham. Property directed reachability for proving absence of concurrent modification errors. In Ahmed Bouajjani and David Monniaux, editors, *Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings*, volume 10145 of *Lecture Notes in Computer Science*, pages 209–227. Springer, 2017. 150

[FGP16] Jean-Christophe Filliâtre, Léon Gondelman, and Andrei Paskevich. The spirit of ghost code. *Formal Methods Syst. Des.*, 48(3):152–174, 2016. 109

[FH07] Harald Fecher and Michael Huth. More precise partition abstractions. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2007. 147

[FISS20] Yotam M. Y. Feldman, Neil Immerman, Mooly Sagiv, and Sharon Shoham. Complexity and information in invariant inference. *Proc. ACM Program. Lang.*, 4(POPL):5:1–5:29, 2020. 6, 37, 78, 152

[FKE+20] Yotam M. Y. Feldman, Artem Khyzha, Constantin Enea, Adam Morrison, Aleksandar Nanevski, Noam Rinetzky, and Sharon Shoham. Proving highly-concurrent traversals correct. *Proc. ACM Program. Lang.*, 4(OOPSLA):128:1–128:29, 2020. 6

[FL01] Cormac Flanagan and K. Rustan M. Leino. 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*, pages 500–517, 2001. 24, 44, 69, 79, 109

[FQ02] Cormac Flanagan and Shaz Qadeer. 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*, pages 191–202, 2002. 3, 24, 72

[FS22] Yotam M. Y. Feldman and Sharon Shoham. Invariant inference with provable complexity from the monotone theory. In *Static Analysis - 29th International Symposium, SAS 2022, Auckland, New Zealand. To appear.*, 2022. 6, 71, 111

[FSSW21] Yotam M. Y. Feldman, Mooly Sagiv, Sharon Shoham, and James R. Wilcox. Learning the boundary of inductive invariants. *Proc. ACM Program. Lang.*, 5(POPL):1–30, 2021. 6, 71

[FSSW22] Yotam M. Y. Feldman, Mooly Sagiv, Sharon Shoham, and James R. Wilcox. Property-directed reachability as abstract interpretation in the monotone theory. *Proc. ACM Program. Lang.*, 6(POPL):1–31, 2022. 6, 111

[FWSS19] Yotam M. Y. Feldman, James R. Wilcox, Sharon Shoham, and Mooly Sagiv. 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*, pages 405–425, 2019. 6, 79, 109, 149

[Gav93] Ricard Gavaldà. On the power of equivalence queries. In John Shawe-Taylor and Martin Anthony, editors, *Proceedings of the First European Conference on Computational Learning Theory, EuroCOLT 1993, London, UK, December 20-22, 1993*, pages 193–203. Oxford University Press, 1993. 4

[GI15] Arie Gurfinkel and Alexander Ivrii. Pushing to the top. In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 65–72, 2015. 22, 123

[GK95] Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. *J. Comput. Syst. Sci.*, 50(1):20–31, 1995. 70

[GLMN13] Pranav Garg, Christof Löding, P Madhusudan, and Daniel Neider. Ice: A robust framework for learning invariants. Technical report, October 2013. 69

[GLMN14] Pranav Garg, Christof Löding, P Madhusudan, and Daniel Neider. Ice: A robust framework for learning invariants. In *Computer Aided Verification*, pages 69–87. Springer, 2014. 2, 5, 7, 9, 11, 23, 24, 42, 43, 48, 58, 59, 66, 68, 69, 70, 78, 90, 108

[GLR15] Roberto Giacobazzi, Francesco Logozzo, and Francesco Ranzato. Analyzing program analyses. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 261–273. ACM, 2015. 109

[GMT08] Sumit Gulwani, Bill McCloskey, and Ashish Tiwari. Lifting abstract interpreters to quantified logical domains. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 235–246. ACM, 2008. 127

[GNMR16] Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. 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, pages 499–512, 2016. 2, 23, 24, 42, 69, 108, 149

[Gol06] Oded Goldreich. On promise problems: A survey. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, pages 254–290, 2006. 45

[GS97] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, pages 72–83, 1997. 3, 24, 72

[GSM16] Arie Gurfinkel, Sharon Shoham, and Yuri Meshman. 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*, pages 338–348, 2016. 149

[Gul12] Sumit Gulwani. Synthesis from examples: Interaction models and algorithms. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 8–14, 2012. 70

[Hak85] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. 70

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. 146

[HB12] Krystof Hoder and Nikolaj Bjørner. Generalized property directed reachability. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pages 157–171, 2012. 3, 22, 40

[HHTW10] Edith Hemaspaandra, Lane A. Hemaspaandra, Till Tantau, and Osamu Watanabe. On the complexity of kings. *Theor. Comput. Sci.*, 411(4-5):783–798, 2010. 133

[HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 232–244, 2004. 23

[HKSS12] Lisa Hellerstein, Devorah Kletenik, Linda Sellie, and Rocco A. Servedio. Tight bounds on proper equivalence query learning of DNF. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 31.1–31.18, 2012. 4, 66, 67

[HPRW96] Lisa Hellerstein, Krishnan Pillaipakkamnatt, Vijay Raghavan, and Dawn Wilkins. How many queries are needed to learn? *J. ACM*, 43(5):840–862, 1996. 4

[IBR+14] Shachar Itzhaky, Nikolaj Bjørner, Thomas W. Reps, Mooly Sagiv, and Aditya V. Thakur. 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*, pages 35–51, 2014. 81

[JGST10] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 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*, pages 215–224, 2010. 2, 11, 43, 69, 70

[JKD+15] Yungbum Jung, Soonho Kong, Cristina David, Bow-Yaw Wang, and Kwangkeun Yi. Automatically inferring loop invariants via algorithmic learning. *Math. Struct. Comput. Sci.*, 25(4):892–915, 2015. 108

[JM07] Ranjit Jhala and Kenneth L. McMillan. Interpolant-based transition relation approximation. *Logical Methods in Computer Science*, 3(4), 2007. 23

[JS17] Susmit Jha and Sanjit A. Seshia. A theory of formal synthesis via inductive learning. *Acta Inf.*, 54(7):693–726, 2017. 2, 11, 43, 66, 70

[JSS14] Bertrand Jeannet, Peter Schrammel, and Sriram Sankaranarayanan. Abstract acceleration of general linear loops. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 529–540, 2014. 7

[KBI+17] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. Property-directed inference of universal invariants or proving their absence. *J. ACM*, 64(1):7:1–7:33, 2017. 3, 149

[KGC14] Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. Smt-based model checking for recursive programs. 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*, pages 17–34, 2014. 3, 40

[KLVW17] Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 719–734. ACM, 2017. 148

[KMW16] Egor George Karpenkov, David Monniaux, and Philipp Wendler. 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*, pages 127–146, 2016. 109

[KPIA20] Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. First-order quantified separators. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 703–717. ACM, 2020. 2, 149

[KS03] Daniel Kroening and Ofer Strichman. Efficient computation of recurrence diameters. In Lenore D. Zuck, Paul C. Attie, Agostino Cortesi, and Supratik Mukhopadhyay, editors, *Verification, Model Checking, and Abstract Interpretation, 4th International Conference, VMCAI 2003, New York, NY, USA, January 9-11, 2002, Proceedings*, volume 2575 of *Lecture Notes in Computer Science*, pages 298–309. Springer, 2003. 147

[KVW14] Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 125–140. Springer, 2014. 148

[KW07] Daniel Kroening and Georg Weissenbacher. 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*, pages 85–89. IEEE Computer Society, 2007. 108

[KW11] Daniel Kroening and Georg Weissenbacher. Interpolation-based software verification with wolverine. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 573–578. Springer, 2011. 3

[LB07] Shuvendu K. Lahiri and Randal E. Bryant. Predicate abstraction with indexed predicates. *ACM Trans. Comput. Log.*, 9(1):4, 2007. 24

[Lip75] Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Commun. ACM*, 18(12):717–721, 1975. 148

[Lit87] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987. 152

[LL90] Kim Guldstrand Larsen and Xinxin Liu. Equation solving using modal transition systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 108–117. IEEE Computer Society, 1990. 135

[LLFB14] Francesco Logozzo, Shuvendu K. Lahiri, Manuel Fähndrich, and Sam Blackshear. Verification modulo versions: towards usable verification. In Michael F. P. O'Boyle and Keshav Pingali, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, pages 294–304. ACM, 2014. 109

[LMN16] Christof Löding, P. Madhusudan, and Daniel Neider. Abstract learning frameworks for synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 167–185, 2016. 24, 44

[LP16] K. Rustan M. Leino and Clément Pit-Claudel. 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*, pages 361–381, 2016. 109

[LPP+17] Vu Le, Daniel Perelman, Oleksandr Polozov, Mohammad Raza, Abhishek Udupa, and Sumit Gulwani. Interactive program synthesis. *CoRR*, 2017. 70

[LQ09] Shuvendu K. Lahiri and Shaz Qadeer. 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*, pages 214–229, 2009. 4, 7, 44, 45, 69

[McM03] Kenneth L. McMillan. Interpolation and sat-based model checking. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, pages 1–13, 2003. xiii, 2, 3, 7, 8, 11, 23, 54, 58, 72, 73, 74, 76, 83, 108, 143

[McM05] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005. 108

[McM06] Kenneth L. McMillan. Lazy abstraction with interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 123–136, 2006. 23

[McM11] Kenneth L. McMillan. Interpolants from Z3 proofs. In Per Bjesse and Anna Slobodová, editors, *International Conference on Formal Methods in Computer-*

Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011, pages 19–27. FMCAD Inc., 2011. 108

[McM18] Kenneth L. McMillan. Interpolation and model checking. In *Handbook of Model Checking*, pages 421–446. 2018. 12, 73

[Min06] Antoine Miné. The octagon abstract domain. *High. Order Symb. Comput.*, 19(1):31–100, 2006. 24, 148

[MRW05] Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. On converting cnf to dnf. *Theoretical Computer Science*, 347(1):325–335, 2005. 102

[Nam07] Kedar S. Namjoshi. 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*, pages 299–313, 2007. 79, 109

[ND12] Srinivas Nidhra and Jagruthi Dondeti. Black box and white box testing techniques - a literature review. *International Journal of Embedded Systems and Applications*, 2:29–50, 06 2012. 69

[NMS+20] Daniel Neider, P. Madhusudan, Shambwaditya Saha, Pranav Garg, and Daejun Park. A learning-based approach to synthesizing invariants for incomplete verification engines. *J. Autom. Reason.*, 64(7):1523–1552, 2020. 2

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. 12, 150

[O'R04] Joseph O'Rourke. Visibility. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 643–663. Chapman and Hall/CRC, 2004. 29, 30

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. 70

[PIS+16] Oded Padon, Neil Immerman, Sharon Shoham, Aleksandr Karbyshev, and Mooly Sagiv. Decidability of inferring inductive invariants. 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*, pages 217–231, 2016. 147

[PMP+16] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and*

Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016, pages 614–630, 2016. 57

[Qui54] WV Quine. Two theorems about truth-functions. *Boletín de la Sociedad Matemática Mexicana*, 10(1–2):64–70, 1954. 13, 25, 98, 147

[Qui86] J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986. 108

[RG13] Jussi Rintanen and Charles Orgill Gretton. Computing upper bounds on lengths of transition sequences. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 2365–2372. IJCAI/AAAI, 2013. 148

[RKG18] Robert Robere, Antonina Kolokolova, and Vijay Ganesh. The proof complexity of SMT solvers. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, pages 275–293, 2018. 70

[RS16] Noam Rinetzky and Sharon Shoham. Property directed abstract interpretation. In Barbara Jobstmann and K. Rustan M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*, volume 9583 of *Lecture Notes in Computer Science*, pages 104–123. Springer, 2016. 16, 147

[RSY04] Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. Symbolic implementation of the best transformer. In *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, Italy, January 11-13, 2004, Proceedings*, pages 252–266, 2004. 8, 24, 69, 148

[RT16] Thomas W. Reps and Aditya V. Thakur. Automating abstract interpretation. In Barbara Jobstmann and K. Rustan M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*, volume 9583 of *Lecture Notes in Computer Science*, pages 3–40. Springer, 2016. 148

[RY20] Xavier Rival and Kwangkeun Yi. *Introduction to Static Analysis: An Abstract Interpretation Perspective*. MIT Press, 2020. 126

[SA16] Rahul Sharma and Alex Aiken. From invariant checking to invariant inference using randomized search. *Formal Methods in System Design*, 48(3):235–256, 2016. 2, 70, 149

[SCIG08] Sriram Sankaranarayanan, Swarat Chaudhuri, Franjo Ivancic, and Aarti Gupta. Dynamic inference of likely data preconditions over predicates by tree learning. In

Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2008, Seattle, WA, USA, July 20-24, 2008, pages 295–306, 2008. 69

[SDDA11] Rahul Sharma, Isil Dillig, Thomas Dillig, and Alex Aiken. Simplifying loop invariant generation using splitter predicates. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 703–719, 2011. 79, 109

[SG09] Saurabh Srivastava and Sumit Gulwani. Program verification using templates over predicate abstraction. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009*, pages 223–234, 2009. 7

[SGF13] Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. Template-based program verification and program synthesis. *STTT*, 15(5-6):497–518, 2013. 3, 7

[SGH+13a] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, Percy Liang, and Aditya V. Nori. 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*, pages 574–592, 2013. 2, 70

[SGH+13b] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, and Aditya V. Nori. Verification as learning geometric concepts. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, pages 388–411, 2013. 2, 69

[SNA12] Rahul Sharma, Aditya V. Nori, and Alex Aiken. Interpolants as classifiers. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 71–87, 2012. 2, 12, 69, 108

[SRW02] Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002. 79, 109

[SS17] Tobias Seufert and Christoph Scholl. Sequential verification using reverse PDR. In Daniel Große and Rolf Drechsler, editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, MBMV 2017, Bremen, Germany, February 8-9, 2017*, pages 79–90. Shaker Verlag, 2017. 21, 148

[SSM04] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constraint-based linear-relations analysis. In *Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings*, pages 53–68, 2004. 7

[SST08] Robert H. Sloan, Balázs Szörényi, and György Turán. On k-term DNF with the largest number of prime implicants. *SIAM J. Discret. Math.*, 21(4):987–998, 2008. 86

[STB+06] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006*, pages 404–415, 2006. 11, 43, 70

[SU02] Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002. 133

[TER12] Aditya V. Thakur, Matt Elder, and Thomas W. Reps. Bilateral algorithms for symbolic abstraction. In Antoine Miné and David Schmidt, editors, *Static Analysis - 19th International Symposium, SAS 2012, Deauville, France, September 11-13, 2012. Proceedings*, volume 7460 of *Lecture Notes in Computer Science*, pages 111–128. Springer, 2012. 148

[TLLR15] Aditya V. Thakur, Akash Lal, Junghee Lim, and Thomas W. Reps. Posthat and all that: Automating abstract interpretation. *Electr. Notes Theor. Comput. Sci.*, 311:15–32, 2015. 8, 24, 69, 148

[Uma01] Christopher Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. Syst. Sci.*, 63(4):597–611, 2001. 133

[Urb15] Caterina Urban. *Static analysis by abstract interpretation of functional temporal properties of programs*. PhD thesis, Paris, Ecole normale supérieure, 2015. 126

[Val84] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. xiii, 4, 13, 27, 78, 149

[VG09] Yakir Vizel and Orna Grumberg. Interpolation-sequence based model checking. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*, pages 1–8, 2009. 23

[VG14] Yakir Vizel and Arie Gurfinkel. Interpolating property directed reachability. 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*, pages 260–276, 2014. 23

[VGM15] Yakir Vizel, Arie Gurfinkel, and Sharad Malik. Fast interpolating BMC. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th*

International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, volume 9206 of *Lecture Notes in Computer Science*, pages 641–657. Springer, 2015. 108

[VGS13] Yakir Vizel, Orna Grumberg, and Sharon Shoham. Intertwined forward-backward reachability analysis using interpolants. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, pages 308–323, 2013. 23

[VGSM17] Yakir Vizel, Arie Gurfinkel, Sharon Shoham, and Sharad Malik. IC3 - flipping the E in ICE. In *Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings*, pages 521–538, 2017. 10, 42, 69

[VNR15] Yakir Vizel, Alexander Nadel, and Vadim Ryvchin. Efficient generation of small interpolants in CNF. *Formal Methods Syst. Des.*, 47(1):51–74, 2015. 108

[Wie87] Douglas H Wiedemann. *Hamming geometry*. PhD thesis, University of Waterloo, 1987. 30, 96, 105, 118

תקציר

הסקת אינווריאנטות המבוססת על שימוש בפותרני ספיקות (SAT) היא גישה בולטת להוכחת נכונות אוטומטית של תכונות בטיחות (safety) של תוכניות, עם אלגוריתמים יעילים מבחינה מעשית, אך שאינם מובנים מבחינה תיאורטית. תזה זה בוחנת שאלות קונספטואליות לגבי אלגוריתמים מבוססי SAT להסקת אינווריאנטות דרך מחקר תיאורטי של הסיבוכיות של אלגוריתמים אלו, בהשראת ובהשפעת תורת הלמידה (learning theory). התוצאות מטילות אור על העקרונות של אלגוריתמים מודרניים מבוססי SAT להסקת אינווריאנטות.

תזה זו מפתחת מודלים מבוססי שאילתות שמתאימים ללמידת אינווריאנטות אינדוקטיביות אשר יכולים למדל אלגוריתמים מרכזיים בתחום; משווה בין הקשיות של למידת אינווריאנטות באמצעות שאילתות ללמידה קלאסית באמצעות שאילתות; מוכיחה חסמים תחתונים מבוססי אינפורמציה על הסקת אינווריאנטות; ובוחנת את הכוח של מודלים בעלי שאילתות עשירות לעומת הכח של מודלים שפותחו קודם לכן בספרות. תזה זו, בנוסף, מפתחת חסמים עליונים על הסיבוכיות של גרסה של אלגוריתם הסקת אינווריאנטות באמצעות אינטרפולנטים (interpolants) בהסקת מחלקה עשירה של אינווריאנטות, ובאמצעות רעיונות מלמידה קלאסית אף מפתחת אלגוריתמים חדשים עם תוצאות סיבוכיות יעילה למחלקות סינטקטיות מעניינות אף יותר של אינווריאנטות, כולל עצי החלטה (decision trees). בנוסף, באמצעות התורה המונוטונית (monotone theory) של בשותי שפותחה במקור ללמידה, תזה זו ממצבת את אלגוריתם ישיגות מונחית-תכונה (property-directed reachability) כסוג של אינטרפטציה אבסטרקטית (abstract interpretation) של תוכניות בתחום אבסטרקטי חדש, ועל ידי כך מסבירה חלקית איך קירוב-יתר (overapproximation) מושג באלגוריתם זה.

אוניברסיטת תל-אביב TEL AVIV UNIVERSITY
הפקולטה למדעים מדויקים ע״ש ריימונד ובברלי סאקלר
בית הספר למדעי המחשב ע״ש בלבטניק

לקראת תיאוריה של למידת אינווריאנטות אינדוקטיביות

חיבור זה הוגש כחלק מהדרישות לקבלת התואר

"דוקטור לפילוסופיה (Ph.D.)"

על ידי

יותם פלדמן

עבודת המחקר בוצעה בהנחייתם של

פרופ' מולי שגיב ופרופ' שרון שוהם-בוכבינדר