SAT-based Model Checking Using Interpolation and IC3

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Yakir Vizel

Sumbitted to the Senate of Technion - Israel Institute of Technology

Iyar, 5774 Haifa May, 2014

This research thesis was done under the supervision of Professor Orna Grumberg in the Department of Computer Science.

First and foremost, I would like to thank my advisor, Professor Orna Grumberg. She has patiently guided me in my journey as a new researcher, assisting whenever possible, while allowing me to grow and practice my academic independence. I had the honor to work with a world leading researcher in the domain of Formal Verification and I am confident that she has given me all the tools required for my future journey and success.

I would like to thank Dr. Sharon Shoham, Dr. Alex Nadel and Dr. Vadim Ryvchin for a fruitful and successful collaboration. I am sure that this experience will be helpful in the future. In addition, I would like to thank Dr. Ziyad Hanna for the support and the oppertunity he gave me by allowing me to bring my research to life in an industrial enviornment. By doing so, he gave me the tools to realize how my research affects real life problems.

I would like to thank my parents, Israel and Sima Vizel, who have taught me to question, explore, persist, and believe in myself. They supported me in every aspect of my studies, from elementary school to graduate school. I also thank my brother and sister, Shay and Shir, for supporting me morally and appriciating my work. They were all an inspiration during my research.

The generous financial help of The Technion is gratefully acknowledged.

List of Publications

- Yakir Vizel, Vadim Ryvchin, Alexander Nadel "Efficient Generation of Small Interpolants in CNF, 25th International Conference on Computer Aided Verification (CAV'13), Saint Petersburg, Russia.
- Yakir Vizel ,Orna Grumberg, Sharon Shoham "Intertwined Forward-Backward Reachability Analysis Using Interpolants", International Conference on Tools and Algorithms for the Construction and Analysis of Systems 2013 (TACAS'13), Rome, Italy.
- Yakir Vizel ,Orna Grumberg, Sharon Shoham "Lazy Abstraction and SAT-based Reachability in Hardware Model Checking", International Conference on Formal Methods in Computer-Aided Design 2012 (FMCAD'12), Cambridge, United Kingdom.
- Yakir Vizel ,Orna Grumberg, "Interpolation-Sequence Based Model Checking", International Conference on Formal Methods in Computer-Aided Design 2009 (FMCAD'09), Austin, Texas.

Table of Contents

List of Figures

List of Tables

Abstract

SAT-based model checking is currently one of the most successful approaches to checking very large systems. In its early days, SAT-based (bounded) model checking was mainly used for bug hunting. The introduction of *interpolation* and *IC3\PDR* enable efficient complete algorithms that can provide full verification as well.

In this thesis, we preset several approaches to enhancing SAT-based model checking. They are all based on iteratively computing an *over-approximation* of the set of reachable system states. They use different mechanisms to achieve scalability and faster convergence (empirically).

The first approach uses *interpolation-sequence*, rather than interpolation, in order to obtain a more precise over-approximation of the set of reachable states and avoids the addition of interpolants into the BMC formula.

The second approach extracts interpolants in both forward and backward manner and exploits them for an intertwined approximated forward and backward reachability analysis. The approach is also mostly local and avoids unrolling of the checked model as much as possible. By that, the size of interpolants is mostly kept small. This results in an efficient and complete SAT-based verification algorithm.

The third approach takes a different direction. It suggests a new method for interpolant computation which is specific for model checking. As a first step, it approximates the interpolant using a proof generated by the SAT solver. The second step transforms the approximated interpolant into a real

interpolant by using the structure of the model checking problem and applying inductive reasoning. This results in an efficient procedure that generates compact interpolants in Conjunctive Normal Form.

The last approach we present integrates lazy abstraction with IC3 in order to achieve scalability. *Lazy abstraction*, originally developed for software model checking, is a specific type of abstraction that allows hiding different model details at different steps of the verification. We find the IC3 algorithm most suitable for lazy abstraction since its state traversal is performed by means of *local* reachability checks, each involving only two consecutive sets. A different abstraction can therefore be applied in each of the local checks.

The techniques presented in this thesis make SAT-based model checking more scalable. The thesis focuses on hardware model checking, but the presented ideas can be extended to other systems as well.

Chapter 1

Introduction

Computerized systems dominate almost every aspect of our lives and their correct behavior is essential. *Model checking* [21, 51, 20] is an automated verification technique for checking whether a given system satisfies a desired property. The system is usually described as a finite-state model in a form of a state transition graph. The specification is given as a temporal logic formula. Unlike testing or simulation based verification, model checking tools are exhaustive in the sense that they traverse *all* behaviors of the system, and either confirm that the system behaves correctly or present a *counterexample*.

Model checking has been successfully applied to verifying hardware and software systems. Its main limitation, however, is the *state explosion problem* which arises due to the huge state space of real-life systems. The size of the model induces high memory and time requirements that may make model checking not applicable to large systems. Much of the research in this area is dedicated to increasing model checking applicability and scalability.

The first significant step in this direction was the introduction of BDDs [12] into model checking. BDD-based *Symbolic Model Checking* (SMC) [13] enabled model checking of real-life hardware designs with a few hundreds of state elements. However, current design blocks with well-defined functionality typically have thousands of state elements and more. To handle designs

of that scale, model checking has been reduced to *satisfiability* (SAT) and SAT-based *Bounded Model Checking* (BMC) [5] has been developed. Its main drawback, however, is its orientation towards "bug-hunting" rather than full verification.

Several approaches have been suggested to remedy the problem and make it applicable for verification. *Induction* [54], *interpolation* [43, 59, 60], *interpolation sequence* [57, 16], *IC3/PDR* [8, 28], and *L-IC3* [58] developed different techniques for SAT-based *Unbounded Model Checking* (UMC), which provide full verification. All techniques are based on finding an inductive invariant that proves the correctness of the verified property. More precisely, most of these techniques explicitly find the inductive invariant by approximating the reachable states in the verified system.

Of these SAT-based unbounded model checking techniques, L-IC3 and [16] also use *Abstraction-refinement* [22], which is another well known methodology for tackling the state-explosion problem. Abstraction hides model details that are not relevant for the checked property. The resulting abstract model is then smaller, and therefore easier to handle by model checking algorithms. *Lazy abstraction* [41, 44], developed for software model checking, is a specific type of abstraction that allows hiding different model details at different steps of the verification.

We now go through challenges in SAT-based model checking and our techniques for improvements.

1.1 Challenges in SAT-based Model Checking

This work focuses on improving Interpolation based model checking as was first introduced in [43], and improving IC3/PDR [8, 28]. These methods compute an over-approximated sets of the system's reachable states while checking that the specification is not violated. The approximation of reach-

able states is done via a form of *generalization*. Basically, generalization is the process of deducing a general fact from knowledge about a single case.

The algorithm that appears in [43], which we refer to as ITP, computes over-approximations of reachable states using Craig interpolants [23]. The interpolants are extracted from a proof of unsatisfiability, generated by a SAT-solver when solving a BMC formula, and they represent an over-approximation of states reachable from the initial states after one transition. The computed over-approximations are used to perform a SAT-based reachability analysis and to fully verify a specification.

In the context of ITP, interpolants are used as a generalization mechanism. By solving a BMC formula of a specific length, and knowing that there is no counterexample of this specific length, interpolants help to generalize this fact into general information about the system's reachable states.

ITP works iteratively and is based on a nested loop. The outer loop controls the bound of the BMC formulas that are checked inside the inner loop. The inner loop iteratively solves a fixed bound BMC formulas. If a BMC formula is unsatisfiable, an interpolant representing an over-approximation of reachable states is extracted[1]. These over-approximations are used to check whether all reachable states in the system have been checked. In case all reachable states are checked we say that a fixpoint is found. In this case, the algorithm terminates concluding that the property holds. Otherwise, the computed interpolant replaces the set of initial states from which the bounded search starts and a new BMC formula is checked (with new initial states). Due to the usage of "new" initial states in the BMC formula, there are cases where such a modified BMC formula is satisfiable. Since the interpolants represent an over-approximation of reachable states, ITP cannot conclude that a counterexample exists when a modified BMC formula is satisfiable. In these cases the inner loop is stopped and the bound is increased by the outer loop.

[1]In these settings interpolants can only be computed for an unsatisfiable formula.

There are, therefore, two inherent weaknesses we try to solve. The first weakness is the sensitivity of ITP to the size of the interpolants. Since interpolants are fed back into the BMC formula (when replacing the initial states with an interpolant), their size may render the BMC problem intractable. The second weakness is the need to constantly increase the bound of the checked BMC formula in order to increase the precision of the computed over-approximations.

In [8] an alternative SAT-based algorithm, called IC3, is introduced. Similarly to ITP, IC3 also computes over-approximations of sets of reachable states. However, while ITP unrolls the model in order to obtain more precise approximations, IC3, improves the precision of the approximations by performing many *local reachability checks* between consecutive time frames that do not require unrolling.

While ITP blindly relies on the SAT-solver to search for a counterexample and generate information about reachable states (in the form of interpolants), IC3 approaches the problem in a different manner. Instead of blindly relying on the SAT-solver, it guides both the search for the counterexample and the computation of reachable states.

Conceptually, IC3 is based on a backward search. Starting with a bad state, it uses a SAT-solver to repeatedly find a one-step predecessor state. Thus, all SAT-queries are local, involving only one instance of the transition relation, and no BMC-unrolling is used. If, when performing the backward search, the bad suffix can be extended all the way to the initial states - a counterexample is found. Otherwise, when a suffix cannot be extended further, a process called *inductive generalization* [9, 7, 8], is used to learn a consequence that blocks the current suffix. More precisely, due to inductive generalization, IC3 generalizes the fact that a single state is unreachable to a consequence representing a set of unreachable states. The conjunction of all such learned consequences is used to represent an over-approximation of reachable states.

While inductive generalization is what enables IC3 to learn strong consequences, it is also its weakness. When inductive generalization fails to generalize a single unreachable state into a set of unreachable states efficiently, IC3 falls into a form of state-enumeration, thus making the algorithm inefficient.

1.2 Our Approaches for SAT-based Model Checking Enhancements

We aim at solving the weaknesses of SAT-based methods as described in the previous section. In the context of interpolation, we deal with interpolants' size by using them or computing them differently, such that their size is either reduced or has less affect on the underlying model checking problem. In the context of IC3, we integrate abstraction into the algorithm in order to make it more efficient. We now go into more detail about the content of this thesis.

In Chapter 3, we present an interpolation-sequence [38, 44] based algorithm. The algorithm, referred to as ISB, combines BMC with interpolation-sequence. ISB works by searching for a counterexample via repeatedly posing BMC queries to a SAT-solver. If a BMC query is satisfied, a counterexample is found. Otherwise, the SAT-solver generates a proof of unsatisability. An interpolation procedure is then used to extract an interpolation sequence. The sequence is used to over-approximate sets of reachable states at different depths. If at any point a fixpoint is reached, ISB terminates indicating the validity of the checked property. Otherwise, the process repeats with another, longer, BMC query. ISB can be viewed as a simple addition to the BMC loop that enables termination.

Unlike ITP, ISB does not require to inject the interpolants into the BMC formula, and thus the size of interpolants does not influence the ability to solve a BMC query. Even though ISB is insensitive to the size of interpolants, it does not outperform ITP on average [16].

In Chapter 4, we present the algorithm *Dual Approximated Reachability*

(DAR), that can be viewed as an evolution of ISB. The algorithms ITP, IC3 and ISB are all based on a froward reachability analysis. DAR adds backward reachability analysis and tightly combines it with forward analysis. By doing so, DAR can execute mostly local reachability checks that are similar in a sense to the reachability queries executed by IC3. DAR can therefore avoid unrolling of the model in most cases. Like ITP and ISB, DAR uses interpolation to compute over-approximations of sets of reachable states. But, due to the fact that it can avoid unrolling, it manages to keep interpolants smaller. In addition, since unrolling is avoided, the queries solved by the SAT-solver are simpler.

In Chapter 5, we introduce a novel technique for computing interpolants specifically suitable to model checking. Our approach uses both the proof of unsatisfiability generated by the SAT-solver and information about the underlying problem. Our method computes interpolants in *Conjunctive Normal Form* (CNF) that are small in size compared to interpolants computed by the traditional method [43]. We evaluate this approach in the context of ITP. We have developed an algorithm, called CNF-ITP, which is similar to ITP but uses our method for interpolant computation. In addition, it exploits the fact that interpolants are given in CNF.

In the last chapter (Chapter 6), we present the algorithm L-IC3, which provides a SAT-based *lazy* abstraction-refinement algorithm based on IC3/PDR. Originally introduced for software verification, lazy abstraction enables to use different abstractions at different steps of verification. To the best of our knowledge, L-IC3 is the first to use lazy abstraction for hardware verification. The local reachability checks that lie in the core of IC3 makes it a natural candidate to be used with lazy abstraction. Thus, L-IC3 is developed on top of IC3.

As was mentioned before, IC3 uses many local reachability checks that only contain one instantiation of the transition relation. L-IC3 uses the *visible variable* abstraction [40], and by that enables the usage of different

sets of visible variables for the different local reachability checks that are used by IC3. In contrast to the generic CEGAR framework [22], L-IC3 is tightly integrated in IC3 and uses IC3-specific features (like the locality of the reachability checks).

Integrating abstraction into IC3 enables us to not only execute more efficient SAT queries (since we use an abstract model), but also makes the process of inductive generalization more effective. This enables L-IC3 to learn stronger consequences when it proves that a given state is unreachable, and by that reduce the effort needed when searching for an inductive invariant. This helps L-IC3 to converge faster.

Chapter 2

Preliminaries

Temporal logic model checking [20] is an automatic approach to formally verifying that a given system satisfies a given specification. The system is often modelled by a finite state transition system and the specification is written in a *temporal logic*. Determining whether a model satisfies a given specification is often based on an exploration of the model's state space in a search for violations of the specification.

In this thesis we focus on hardware. As such we consider finite state transition systems defined over Boolean variables, as follows.

Definition 2.0.1. A *finite transition system* or a *model* is a triple $M = (V, INIT, TR)$ where V is a set of boolean variables, $INIT(V)$ is a formula over V, describing the initial states, and $TR(V, V')$ is a formula over V and the next-state variables $V' = \{v' | v \in V\}$, describing the transition relation.

Throughout the thesis we assume that for a given model M, the transition relation TR is total.

The set of Boolean variables of M induces a set of states $S = \{0, 1\}^{|V|}$, where each state $s \in S$ is given by a valuation of the variables in V. A formula over V (resp. V, V') represents the set of states (resp. pairs of states) obtained by its satisfying assignments. With abuse of notation we

will refer to a formula η over V as a set of states and therefore use the notion $s \in \eta$ for states represented by η. Similarly for a formula η over V, V', we will sometimes write $(s, s') \in \eta$.

The formula $\eta[V \leftarrow V']$, or η' in short, is identical to η except that each variable $v \in V$ is replaced with v'. In the general case V^i is used to denote the variables in V after i time units (thus, $V^0 \equiv V$). Let η be a formula over V^i, the formula $\eta[V^i \leftarrow V^j]$ is identical to η except that for each variable $v \in V$, v^i is replaced with v^j. Throughout the paper we denote the value *false* as $\bot$ and the value *true* as $\top$. For a propositional formula η we use $\mathrm{Vars}(\eta)$ to denote the set of all variables appearing in η. For a set of formulas $\{\eta_1, \ldots, \eta_n\}$ $\mathrm{Vars}(\eta_1, \ldots, \eta_n)$ denotes the variables appearing in $\eta_1, \ldots, \eta_n$. That is, $\mathrm{Vars}(\eta_1, \ldots, \eta_n) = \mathrm{Vars}(\eta_1) \cup \ldots \cup \mathrm{Vars}(\eta_n)$.

A *path* in M is a sequence of states $\pi = s_0, s_1, \ldots$ such that for all $0 \leq i \leq |\pi|$, $s_i \in S$ and $(s_i, s_{i+1}) \in TR$. The length of a path is denoted by $|\pi|$. If π is infinite then $|\pi| = \infty$. If $\pi = s_0, s_1, \ldots, s_n$ then $|\pi| = n$. A path is an *initial path* when $s_0 \in INIT$. We sometimes refer to a prefix of a path as a path as well.

We use the following notation to describe a path in M of length $j - i$ by means of propositional formula:

Formula 1. $path^{i,j} = TR(V^i, V^{i+1}) \wedge \ldots \wedge TR(V^{j-1}, V^j)$

where $0 \leq i < j$.

To describe a path of length k starting at the initial states, we will use: $INIT(V^0) \wedge \mathrm{path}^{0,k}$.

A formula in *Linear Temporal Logic* (LTL) [49, 20] is of the form $\mathbf{A}f$ where f is a path formula. A model M satisfies an LTL property $\mathbf{A}f$ if all infinite initial paths in M satisfy f. If there exists an infinite initial path not satisfying f, this path is defined to be a *counterexample*.

In this thesis we consider a subset of LTL formulas of the form $\mathbf{AG}\,p$, where p is a propositional formula. $\mathbf{AG}\,p$ is true in a model M if along every initial infinite path all states satisfy the proposition p. In other words, all

states in M that are reachable from an initial state satisfy p. This does not restrict the generality of the suggested methods since model checking of liveness properties can be reduced to handling safety properties [3]. Further, model checking of safety properties can be reduced to handling properties of the form $\mathbf{AG}\, p$ [39].

As was mentioned before, the *model checking problem* is the problem of determining whether a given model satisfies a given property. For properties of the form $\mathbf{AG}\, p$ this can be done by traversing the set of all states reachable from the initial states, called *reachable states* in short. Let M be a model, *Reach* be the set of reachable states in M, and $f = \mathbf{AG}\, p$ be a property. If for every $s \in \mathit{Reach}$, $s \models p$ then the property holds in M. On the other hand, if there exists a state $s \in \mathit{Reach}$ such that $s \models \neg p$ then there exists an initial path $\pi = s_0, s_1, \ldots, s_n$ such that $s_n = s$. The path π is a *counterexample* for the property f.

Model checking has been successfully applied to hardware verification, and is emerging as an industrial standard tool for the verification of hardware designs. The main technical challenge in model checking, however, is the *state explosion* problem which occurs if the system is a composition of several components or if the system variables range over large domains.

2.1 Satisfiability

Many problems, including some versions of model checking, can naturally be translated into the *satisfiability* problem of the propositional calculus. The satisfiability problem is known to be NP-complete. Nevertheless, modern SAT-solvers, developed in recent years, can check satisfiability of formulas with several thousands of variables within a few seconds. SAT-solvers such as Grasp [55], Chaff [48], MiniSAT [29], Glucose [1], and many others, are based on sophisticated learning techniques and data structures that accelerate and increase the efficiency of the search for a satisfying assignment, if it exists.

Definition 2.1.1 (Conjunctive Normal Form)**.** Given a set U of Boolean variables, a literal l is a variable $u \in U$ or its negation and a *clause* is a disjunction of literals. A formula F in *Conjunctive Normal Form* (CNF) is a conjunction of clauses.

With abuse of notation, we sometimes refer to a clause as a set of literals and to a CNF formula as a set of clauses.

A SAT-solver is a complete decision procedure that given a propositional formula, determines whether the formula is *satisfiable* or *unsatisfiable*. Most SAT-solvers assume a formula in CNF. A CNF formula is *satisfiable* if there exists a *satisfying assignment* for which every clause in the set is evaluated to $\top$. If the clause set is satisfiable then the SAT solver returns a satisfying assignment for it. If it is not satisfiable (unsatisfiable), meaning, it has no satisfying assignment, then modern SAT-solvers produce a *resolution refutation* comprising the proof of unsatisfiability [62, 30, 45]. The proof of unsatisfiability has many useful applications. We will introduce one of them in a following section.

2.2 Bounded Model Checking

We now describe how to exploit satisfiability for bounded model checking of properties of the form $\mathbf{AG}\, p$, where p is a propositional formula.

Bounded model checking (BMC) [5] is an iterative process for checking properties of a given model up to a given bound. Let M be a model and $f = \mathbf{AG}\, p$ be the property to be verified. Given a bound k, BMC either finds a counterexample of length k or less for f in M, or concludes that there is no such counterexample. In order to search for a counterexample of length k the following propositional formula is built:

Formula 2. $\varphi^k_M(f) = \mathit{INIT}(V^0) \wedge \mathrm{path}^{0,k} \wedge (\neg p(V^k))$

$\varphi^k_M(f)$ is then passed to a SAT-solver which searches for a satisfying

```
1: function BMC(M,f,k)
2:     i := 0
3:     while i ≤ k do
4:         build φ^i_M(f)
5:         result = SAT(φ^i_M(f))
6:         if result == true then
7:             return cex      // returning the counterexample
8:         end if
9:         i = i + 1
10:    end while
11:    return No cex for bound k
12: end function
```

Figure 2.1: Bounded model checking

assignment. If there exists a satisfying assignment for $\varphi^k_M(f)$ then the property $\mathbf{AG}\, p$ is violated, since there exists a path of M of length k violating the property. In order to conclude that there is no counterexample of length k or less, BMC iterates all lengths from 0 up to the given bound k. At each iteration a SAT procedure is invoked.

When M and f are obvious from the context we omit them from the formula $\varphi^k_M(f)$ denoting it as φ^k. The BMC algorithm is described in Figure 2.1.

The main drawback of this approach is its incompleteness. It can only guarantee that there is no counterexample of size smaller or equal to k. It cannot guarantee that there is no counterexample of size greater than k.

Thus, this method is mainly suitable for refutation. Verification is obtained only if the bound k exceeds the length of the longest path among all shortest paths from an initial state to some state in M. In practice, it is hard to compute this bound and even when known, it is often too large to handle. As mentioned before, several methods for full verification with SAT have been suggested, such as induction [54], ALL-SAT [46, 31], interpolation [43, 47, 57], and Property Directed Reachability (PDR/IC3) [8, 28, 58].

In the rest of the thesis we will focus on SAT-based verification with interpolation and PDR.

2.3 Interpolation

In this section we introduce two notions, *interpolation* [23] and *interpolation-sequence* [38] that, when combined with BMC, can provide full program verification.

Given a pair of unsatisfiable propositional formulas $A(X, Y)$ and $B(Y, Z)$, where X, Y and Z are sets of Boolean variables, an interpolant $I(Y)$ is a formula that fulfills the following properties: $A(X, Y) \Rightarrow I(Y)$; $I(Y) \wedge B(Y, Z)$ is unsatisfiable; and $I(Y)$ is a formula over the common variables of $A(X, Y)$ and $B(Y, Z)$ [23]. Modern SAT-solvers are capable of generating an unsatisfiability proof of an unsatisfiable formula. The proof is in the form of a resolution refutation [61, 30, 45]. It is possible to compute an *interpolant* from a resolution refutation of $A(X, Y) \wedge B(Y, Z)$ [50, 43].

Definition 2.3.1. Let (A, B) be a pair of formulas such that $A \wedge B \equiv \bot$. The *interpolant* for (A, B) is a formula I such that:

- $A \Rightarrow I$.
- $I \wedge B \equiv \bot$.
- $\mathrm{Vars}(I) \subseteq \mathrm{Vars}(A) \cap \mathrm{Vars}(B)$.

The interpolant can be viewed as the part of A that is sufficient to contradict B. Note that different proofs yield different interpolants.

A similar notion can be defined when we have a sequence of formulas whose conjunction is unsatisfiable.

Definition 2.3.2. Let $\Gamma = \langle A_1, A_2, \ldots, A_n \rangle$ be a sequence of formulas such that $\bigwedge \Gamma \equiv \bot$. That is $\bigwedge \Gamma = A_1 \wedge \ldots \wedge A_n$ is unsatisfiable. An *interpolation-sequence* for Γ is a sequence $\langle \mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n \rangle$ such that:

1. $\mathcal{I}_0 \equiv \top$ and $\mathcal{I}_n \equiv \bot$
2. For every $0 \leq j < n$ it holds that $\mathcal{I}_j \wedge A_{j+1} \Rightarrow \mathcal{I}_{j+1}$
3. For every $0 < j < n$ it holds that $\text{Vars}(\mathcal{I}_j) \subseteq \text{Vars}(A_1, \ldots, A_j) \cap \text{Vars}(A_{j+1}, \ldots, A_n)$

Computing an interpolation-sequence for a sequence of formulas is done in the following way: given a proof of unsatisfiability π, for each $\mathcal{I}_i$, $0 < i < n$, the sequence of formulas is partitioned in a different way such that $\mathcal{I}_i$ is the interpolant for the formulas $A(i) = \bigwedge_{j=1}^{i} A_j$ and $B(i) = \bigwedge_{j=i+1}^{n} A_j$, obtained based on π. In fact, all interpolants $\mathcal{I}_i$ in the sequence can be computed efficiently at once, by a single traversal of a given proof of unsatisfiability [57].

Before proving the above, we provide some resolution-related definitions. The *resolution rule* states that given clauses $\alpha_1 = \beta_1 \vee v$ and $\alpha_2 = \beta_2 \vee \neg v$, where β_1 and β_2 are also clauses, one can derive the clause $\alpha_3 = \beta_1 \vee \beta_2$. Application of the resolution rule is denoted by $\alpha_3 = \alpha_1 \otimes^v \alpha_2$. v is called the *pivot* variable.

Definition 2.3.3 (Resolution Derivation)**.** A *resolution derivation* of a target clause α from a CNF formula $G = \{\alpha_1, \alpha_2, \ldots, \alpha_q\}$ is a sequence $\pi = (\alpha_1, \alpha_2, \ldots, \alpha_q, \alpha_{q+1}, \alpha_{q+2}, \ldots, \alpha_p \equiv \alpha)$, where each clause α_i for $i \leq q$ is *initial* and α_i for $i > q$ is *derived* by applying the resolution rule to α_j and α_k, for some $j, k < i$.

A resolution derivation π can naturally be conceived of as a directed acyclic graph (DAG) whose vertices correspond to all the clauses of π and in which there is an edge from a clause α_j to a clause α_i iff $\alpha_i = \alpha_j \otimes^v \alpha_k$ for some k. A clause $\beta \in \pi$ is a *parent* of $\alpha \in \pi$ iff there is an edge from β to α.

Definition 2.3.4 (Resolution Refutation)**.** A resolution derivation π of the empty clause $\square$ from a CNF formula G is called the *resolution refutation* of G.

An interpolant can be produced out of a resolution refutation [43]. We define $\langle c|B\rangle = \bigvee\{l \in c|\text{var}(l) \in \text{Vars}(B)\}$ as the projection clause achieved by removing all literals such that their variable does not appear in B. We can also generalize the projection to sets of variables.

Definition 2.3.5 (Partial Interpolant)**.** Let (A, B) be a pair of clause sets. Given a proof of unsatisfiability π for $A \cup B$ and a clause c in the proof $(c \in \pi)$, the *partial interpolant* $p(c)$ is defined as:

- if c is a leaf in π, i.e. $c \in A \cup B$:
  - if $c \in A$ then $p(c) = \langle c|B\rangle$
  - else $p(c) = \top$
- else, let c_1, c_2 be parent nodes of c and let v be their pivot variable, i.e. $c = c_1 \otimes^v c_2$:
  - if v is local to A then $p(c) = p(c_1) \vee p(c_2)$
  - else $p(c) = p(c_1) \wedge p(c_2)$

If c is a clause, and l is a literal that does not appear in c, we write $\langle l, c\rangle$ to indicate the clause that results from adding l as a literal to the clause c.

Definition 2.3.6. Clause interpolation sequence has the form $(A_1, \ldots, A_n) \vdash \langle c\rangle[\varphi_1 \ldots, \varphi_{n-1}]$ where $(A_1, \ldots, A_n)$ are clause sets, c is a clause, and $\varphi_1, \ldots, \varphi_{n-1}$ are formulas. It is said to be valid when $\varphi_i \wedge A_{i+1} \Rightarrow (\varphi_{i+1} \vee \langle c|A_{i+1}, \ldots, A_n\rangle)$.

Let us define the pair $(A(i), B(i))$ for $1 \leq i < n$ such that $A(i) = A_1 \wedge \ldots \wedge A_i$ and $B(i) = A_{i+1} \wedge \ldots \wedge A_n$. If the sequence $(A_1, \ldots, A_n)$ is unsatisfiable (i.e. the conjunction of all formulas is unsatisfiable), then every such pair is unsatisfiable. More precisely, $A(i) \wedge B(i)$ is unsatisfiable for $1 \leq i < n$. Given a resolution refutation π for the sequence $(A_1, \ldots, A_n)$, let us define $p_i(c)$ for $c \in \pi$ as the partial interpolant with respect to the pair $(A(i), B(i))$ (Definition 2.3.5).

Lemma 2.3.7. *Let $(A_1, \ldots, A_n)$ be an inconsistent sequence of formulas and let π be a proof of unsatisfiability. For a clause $c \in \pi$ $(A_1, \ldots, A_n) \vdash \langle c \rangle [p_1(c), \ldots, p_{n-1}(c)]$ is valid.*

Proof. Let π be a resolution refutation for the sequence $(A_1, \ldots, A_n)$. We prove by induction on the structure of π. The base case is where c is a root clause of π, meaning, $c \in A_1 \cup \ldots \cup A_n$. We therefore want to show that $p_i(c) \wedge A_{i+1} \Rightarrow p_{i+1}(c) \vee \langle c | A_{i+1}, \ldots, A_n \rangle$. Let us assume that $c \in A_j$:

- $j \leq i$: We need to show that $\langle c | A_{i+1}, \ldots, A_n \rangle \wedge A_{i+1} \Rightarrow \langle c | A_{i+2}, \ldots, A_n \rangle \vee \langle c | A_{i+1}, \ldots, A_n \rangle$, which is equivalent to $\langle c | A_{i+1}, \ldots, A_n \rangle \wedge A_{i+1} \Rightarrow \langle c | A_{i+1}, \ldots, A_n \rangle$. Clearly, this holds $(\psi \wedge \eta \Rightarrow \psi)$.

- $j = i + 1$: We need to show that $\top \wedge A_{i+1} \Rightarrow \langle c | A_{i+2}, \ldots, A_n \rangle \vee \langle c | A_{i+1}, \ldots, A_n \rangle$. Since $c \in A_{i+1}$, $\langle c | A_{i+1}, \ldots, A_n \rangle$ is equivalent to c. Thus, we need to show that $A_{i+1} \Rightarrow c$. This trivially holds since $c \in A_{i+1}$.

- $j > i + 1$: We need to show that $\top \wedge A_{i+1} \Rightarrow \top \vee \langle c | A_{i+1}, \ldots, A_n \rangle$, which is equivalent to $A_{i+1} \Rightarrow \top$ - trivially holds.

For the induction step, let $\langle v, c_1 \rangle$ and $\langle \neg v, c_2 \rangle$ be children of c and let v be the pivot variable. We therefore have the following assumptions:

- $p_i(\langle v, c_1 \rangle) \wedge A_{i+1} \Rightarrow p_{i+1}(\langle v, c_1 \rangle) \vee \langle v \vee c_1 | A_{i+1}, \ldots, A_n \rangle$

- $p_i(\langle \neg v, c_2 \rangle) \wedge A_{i+1} \Rightarrow p_{i+1}(\langle \neg v, c_2 \rangle) \vee \langle \neg v \vee c_2 | A_{i+1}, \ldots, A_n \rangle$

Considering Definition 2.3.5, if $v \in \text{Vars}(A_{i+2}, \ldots, A_n)$ then $p_j(c) = p_j(\langle v, c_1 \rangle) \wedge p_j(\langle \neg v, c_2 \rangle)$ for $j \in \{i, i+1\}$, otherwise if $v \in \text{Vars}(A_{i+1}, \ldots, A_n)$ then $p_i(c) = p_i(\langle v, c_1 \rangle) \wedge p_i(\langle \neg v, c_2 \rangle)$ and $p_{i+1}(c) = p_{i+1}(\langle v, c_1 \rangle) \vee p_{i+1}(\langle \neg v, c_2 \rangle)$. In all other cases $p_j(c) = p_j(\langle v, c_1 \rangle) \vee p_j(\langle \neg v, c_2 \rangle)$. We consider different cases according to v.

- $v \in \text{Vars}(A_{i+2}, \ldots, A_n)$: We need to show that $(p_i(\langle v, c_1\rangle) \wedge p_i(\langle \neg v, c_2\rangle)) \wedge A_{i+1} \Rightarrow (p_{i+1}(\langle v, c_1\rangle) \wedge p_{i+1}(\langle \neg v, c_2\rangle)) \vee \langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$. Let us assume that the above does not hold. There is therefore an assignment such that $(p_i(\langle v, c_1\rangle) \wedge p_i(\langle \neg v, c_2\rangle)) \wedge A_{i+1}$ is evaluated to $\top$ and $(p_{i+1}(\langle v, c_1\rangle) \wedge p_{i+1}(\langle \neg v, c_2\rangle)) \vee \langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$ is evaluated to $\bot$. Thus, $p_i(\langle v, c_1\rangle)$ and $p_i(\langle \neg v, c_2\rangle)$ are evaluated to $\top$. By our induction hypothesis $p_{i+1}(\langle v, c_1\rangle) \vee \langle v \vee c_1 | A_{i+1}, \ldots, A_n\rangle$ and $p_{i+1}(\langle \neg v, c_2\rangle) \vee \langle \neg v \vee c_2 | A_{i+1}, \ldots, A_n\rangle$ are evaluated to $\top$. Due to our assumption, we know that $p_{i+1}(\langle v, c_1\rangle)$ and $p_{i+1}(\langle \neg v, c_2\rangle)$ are both $\bot$. By that, $\langle v \vee c_1 | A_{i+1}, \ldots, A_n\rangle$ and $\langle \neg v \vee c_2 | A_{i+1}, \ldots, A_n\rangle$ are $\top$. Without loss of generality, let us assume that v is evaluated to $\bot$. By that we get that $\langle c_1 | A_{i+1}, \ldots A_n\rangle$ is evaluated to $\top$. This contradicts our assumption that $\langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$ is evaluated to $\bot$.

- $v \in \text{Vars}(A_{i+1}, \ldots, A_n)$: We need to show that $(p_i(\langle v, c_1\rangle) \wedge p_i(\langle \neg v, c_2\rangle)) \wedge A_{i+1} \Rightarrow (p_{i+1}(\langle v, c_1\rangle) \vee p_{i+1}(\langle \neg v, c_2\rangle)) \vee \langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$. This case is proved in a similar manner to the previous case.

- Otherwise: We need to show that $(p_i(\langle v, c_1\rangle) \vee p_i(\langle \neg v, c_2\rangle)) \wedge A_{i+1} \Rightarrow (p_{i+1}(\langle v, c_1\rangle) \vee p_{i+1}(\langle \neg v, c_2\rangle)) \vee \langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$. Let us assume to the contrary, that it does not hold. This means that there exists an assignment such that $(p_i(\langle v, c_1\rangle) \vee p_i(\langle \neg v, c_2\rangle)) \wedge A_{i+1}$ is evaluated to $\top$ and $(p_{i+1}(\langle v, c_1\rangle) \vee p_{i+1}(\langle \neg v, c_2\rangle)) \vee \langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$ is evaluated to $\bot$. Without loss of generality, let us assume that $p_i(\langle v, c_1\rangle)$ is evaluated to $\top$, then by the induction hypothesis $\langle v \vee c_1 | A_{i+1}, \ldots, A_n\rangle$ is also evaluated to $\top$. Since we assume that $\langle c_1 \vee c_2 | A_{i+1}, \ldots, A_n\rangle$ is evaluated to $\bot$ (our contradictory assumption), we get that $\langle c_1 | A_{i+1}, \ldots, A_n\rangle$ is evaluated to $\bot$. But, since $v \notin \text{Vars}(A_{i+1}, \ldots, A_n)$ we get that $\langle c_1 | A_{i+1}, \ldots, A_n\rangle = \langle v \vee c_1 | A_{i+1}, \ldots, A_n\rangle$. This leads to a contradiction.

□

Theorem 2.3.8. *Let* $\Gamma = \langle A_1, A_2, \ldots, A_n \rangle$ *be a sequence of formulas such that* $\bigwedge \Gamma \equiv \bot$ *and let* π *be a proof of unsatisfiability for* $\bigwedge \Gamma$*. For every* $1 \leq i < n$ *let us define* $A(i) = A_1 \wedge \ldots \wedge A_i$ *and* $B(i) = A_{i+1} \wedge \ldots \wedge A_n$*. Let* $\mathcal{I}_i$ *be the interpolant for the pair* $(A(i), B(i))$ *extracted using* π *then the sequence* $\langle \top, \mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{n-1}, \bot \rangle$ *is an interpolation sequence for* Γ*.*

Proof. The proof is immediate from Lemma 2.3.7. □ □

2.4 Interpolation Based Model Checking (ITP)

In [43], interpolation has been suggested for the first time in order to obtain a SAT-based model checking algorithm for full verification. Before going into details, and in order to better understand the algorithm and the motivation behind it, we first review some basic concepts of Symbolic Model Checking (SMC).

2.4.1 Symbolic Model Checking

SMC performs *forward reachability analysis* by computing sets of reachable states S_j, where j is the number of transitions needed to reach a state in S_j when starting from an initial state. More precisely, $S_0(V) = INIT(V)$ and for every $j \geq 1$, $S_{j+1}(V') = \exists V(S_j(V) \wedge TR(V, V'))$. The computation of S_{j+1} is referred to as an *image* operation on the set S_j. Once S_j is computed, if it contains states violating p (recall that $f = AGp$), a counterexample of length j is found and returned. Otherwise, if for $j \geq 1$ $S_j \subseteq \bigcup_{i=0}^{j-1} S_i$ then a *fixpoint* has been reached, meaning that all reachable states have been found already. If no reachable state violates the property then the algorithm concludes that $M \models f$.

2.4.2 ITP Detailed

The algorithm, referred to as *Interpolation Based Model Checking* (ITP), combines BMC and Interpolation [23].

As we have seen, BMC alone is only sound and not complete. In order to be able to determine if $M \models f$, current SAT-based model checking algorithms are based on a computation that over-approximates the reachable states of M. We use the notion of *Reachability Sequence*:

Definition 2.4.1. A *Forward Reachability Sequence* (FRS) of length k with respect to a model M and a property $\mathbf{AG}\, p$, denoted $\bar{F}_{[k]}(M, p)$, is a sequence $\langle F_0, \ldots, F_k \rangle$ of propositional formulas over V such that the following holds:

- $F_0 = INIT$
- $F_i \wedge TR \Rightarrow F'_{i+1}$ for $0 \leq i < k$
- $F_i \Rightarrow p$ for $0 \leq i \leq k$

A reachability sequence $\bar{F}_{[k]}$ is said to be *monotonic* (MFRS) when $F_i \Rightarrow F_{i+1}$ for $0 \leq i < k$.

Recall that the formula F'_{i+1} is equivalent to $F_{i+1}[V \leftarrow V']$, and that implication between formulas corresponds to inclusion between the set of states represented by the formulas. Thus, for non-monotonic reachability sequence, the set of states represented by F_i over-approximates the states reachable from $INIT$ in exactly i steps. When $\bar{F}_{[k]}$ is monotonic F_i represents all the states that are reachable from $INIT$ in i steps *or less*. We refer to i as *time frame (or frame) i*. When M, p and k are clear from the context we omit them and write $\bar{F}$.

Definition 2.4.2 (Fixpoint)**.** [1] Let $\bar{F}$ be a FRS of length n. We say that $\bar{F}$ is at *fixpoint* if there exists $0 < k \leq n$ s.t. $F_k \Rightarrow \bigvee_{i=0}^{k-1} F_i$.

[1] Note that this is an abuse of the fixpoint notation.

ITP uses interpolants to compute a forward reachability sequence (Definition 2.4.1). The algorithm concludes that the property holds when a fixpoint is reached during the computation of the reachable states and none of the computed states violates the property.

Informally, we will use the notion of *fixpoint* when we can conclude that all *reachable* states in the model have been visited[2]. Using a FRS enables us to determine wether a fixpoint has been reached or not.

Theorem 2.4.3. *Let $\bar{F}$ be a FRS of length n for M and AGp. If $\bar{F}$ is at fixpoint then $M \models AGp$.*

Proof. Suppose $\bar{F}$ is at fixpoint, i.e., there exists $0 < k \leq n$ s.t. $F_k \Rightarrow \bigvee_{i=0}^{k-1} F_i$. Denote by R the set of all states reachable from $INIT$ (in any number of steps). Recall that $F_i \Rightarrow p$ for every $0 \leq i \leq n$, which ensures $\bigvee_{i=0}^{k-1} F_i \Rightarrow p$. It therefore suffices to show that $R \Rightarrow \bigvee_{i=0}^{k-1} F_i$ in order to conclude that $R \Rightarrow p$ and thus $M \models AGp$.

We show that $R \Rightarrow \bigvee_{i=0}^{k-1} F_i$. Assume to the contrary that there exists a state in R which is not in $\bigvee_{i=0}^{k-1} F_i$. Consider such a state s whose distance from $INIT$ is shortest. Let s_p be the predecessor of s along a shortest path from $INIT$ to s. The distance of s_p from $INIT$ is shorter than the distance of s. Thus, since s is the closest to $INIT$ which is not in $\bigvee_{i=0}^{k-1} F_i$, it has to be that $s_p \in \bigvee_{i=0}^{k-1} F_i$, which means there exists some $0 \leq j \leq k-1$ s.t. $s_p \in F_j$. Since $F_j \wedge TR \Rightarrow F'_{j+1}$ and s is a successor of s_p, this implies that $s \in F_{j+1}$ where $1 \leq j+1 \leq k$. Therefore, $s \in \bigvee_{i=0}^{k} F_i$. Since $F_k \Rightarrow \bigvee_{i=0}^{k-1} F_i$, we have that $\bigvee_{i=0}^{k-1} F_i \equiv \bigvee_{i=0}^{k} F_i$. We conclude that $s \in \bigvee_{i=0}^{k-1} F_i$, in contradiction. □

The following definition is useful in explaining the interpolation based algorithm. Recall that the verified property is of the form $f = \mathbf{AG}\, p$.

[2]Since an over-approximated sets of reachable states are computed, the computed sets are not monotonic. Therefore, a monotonic function g for which the existence of a fixpoint is guaranteed cannot be defined.

```
1: function ITP(M,p)
2:     if INIT ∧ ¬p == SAT then
3:         return cex
4:     end if
5:     k = 1
6:     while true do
7:         result = ComputeReachable(M, p, k)
8:         if result == fixpoint then
9:             return Valid
10:        else if result == cex then
11:            return cex
12:        end if
13:        k = k + 1
14:    end while
15: end function
```

Figure 2.2: Interpolation-Based Model Checking (ITP)

Definition 2.4.4. For a set of states X, a natural number $N \in \mathbb{N}$ and $1 \leq j \leq N$, X is a *S_j-approximation* w.r.t N if the following two conditions hold: $S_j \subseteq X$ and there is no path of length $(N - j)$ or less violating p, starting from a state $s \in X$. We write $S_j \preceq_N X$ to denote that X is a S_j-approximation w.r.t N.

Note that the formula φ^k is used in BMC to represent a counterexample of length exactly k. This formula can be modified to represent a counterexample of length l for $1 \leq l \leq k$. We denote this formula by $\varphi^{1,k}$ and write $BMC(M, f, 1, k)$ when BMC runs on $\varphi^{1,k}$.

Formula 3. $\varphi^{1,k} = INIT(V^0) \wedge TR(V^0, V^1) \wedge \text{path}^{1,k}(V^1, \ldots, V^k) \wedge (\bigvee_{j=1}^{k} \neg p(V^j))$

Consider the following partitioning for $\varphi^{1,k}$:

- $A = INIT(V^0) \wedge TR(V^0, V^1)$
- $B = \bigwedge_{i=1}^{k-1} TR(V^i, V^{i+1}) \wedge (\bigvee_{j=1}^{k} \neg p(V^j))$.

16: **function** ComputeReachable(M,p, k)
17: $R_0^k = INIT$, $J_0^k = INIT$, $n = 1$
18: **if** $J_0^k \wedge \text{path}^{0,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k)) == \text{SAT}$ **then**
19: **return** *cex*
20: **end if**
21: **repeat**
22: $A = J_{n-1}^k(V^0) \wedge TR(V^0, V^1)$
23: $B = \text{path}^{1,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k))$
24: $J_n^k = \text{GetInterpolant}(A, B)$
25: **if** $J_n^k \Rightarrow R_{n-1}^k$ **then**
26: **return** $fixpoint$
27: **end if**
28: $R_n^k = R_{n-1}^k \vee J_n^k$
29: $n = n + 1$
30: **until** $J_{n-1}^k \wedge \text{path}^{0,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k)) == \text{SAT}$
31: **end function**

Figure 2.3: Inner loop of ITP

Clearly $\varphi^{1,k} \equiv A \wedge B$. Assume that $\varphi^{1,k}$ is unsatisfiable. By the interpolation theorem [23], there exists an interpolant J_1^k which, by Definition 2.3.1, has the following properties:

- J_1^k is defined over the variables of $\text{Vars}(A) \cap \text{Vars}(B)$, namely, V^1.
- $A \Rightarrow J_1^k$. Hence, $S_1 \subseteq J_1^k$.
- $J_1^k(V^1) \wedge B$ is unsatisfiable. This means that there is no path of length $k-1$ or less, starting from J_1^k, which violates p.

By the above we get that $S_1 \preceq_k J_1^k$. At this point, we get the reachability sequence $\langle INIT, J_1^k \rangle$. We can now proceed by replacing the initial states of M with the computed interpolant J_1^k. BMC is reinvoked with the same bound k and with the modified model $M' = (V, J_1^k[V^1 \leftarrow V], TR)$ in which the initial states are J_1^k. A new interpolant J_2^k is then extracted. J_2^k satisfies $S_2 \preceq_{k+1} J_2^k$. The reachability sequence is then updated and contains a new element $\langle INIT, J_1^k, J_2^k \rangle$.

It is important to notice that J_1^k now satisfies $S_1 \preceq_{k+1} J_1^k$ since the BMC run on M' did not find a counterexample of length k starting from a state in J_1^k. In the general case we replace $INIT$ with J_i^k and get J_{i+1}^k. By that, at the end of the i-th iteration, for a given bound k, the reachability sequence is $\langle INIT, J_1^k, J_2^k, \ldots, J_i^k \rangle$.

Figure 2.3 presents, for a given bound k, the computation of an over-approximated set of reachable states. Note that after L iterations of the main loop in CheckReachable we get L interpolants and for every $1 \leq i \leq L$, $S_i \preceq_{k+L} J_i^k$. All computed states are collected in R. If at any iteration, the interpolant J is contained in R, then all reachable states have been found with no violation of f. CheckReachable then returns "$fixpoint$".

On the other hand, if a counterexample is found on a modified model, then ComputeReachable(M,f,k) is aborted, the reachability sequence is discarded, and ComputeReachable(M,f,$k+1$) is initiated. CheckReachable now tries to construct a new reachability sequence. Recall that the counterexample has been obtained on an over-approximated set of states and therefore might not represent a real counterexample in the original model. In case a real counterexample exists, it will be found during a BMC run on the original model M for a larger bound. The complete ITP algorithm appears in Figure 2.2

Chapter 3

Exploiting Interpolation-Sequence in Model Checking

In this section we present a SAT-based algorithm for full verification (sometimes also called *unbounded model checking* (UMC)), which combines BMC and interpolation-sequence [57]. BMC is used to search for counterexamples while the interpolation-sequence is used to produce over-approximated sets of reachable states and to check for termination.

Interpolation-sequence has been introduced and used in [38] and [44]. In [38] it is used for computing an abstract model based on predicate abstraction for software model checking. In [44] interpolation-sequence is used for software model checking and lazy abstraction and is applied to individual execution paths in the control flow graph. The method presented in this section exploits interpolation-sequence in a different manner. In particular, it is applied to the whole model for imitating *symbolic model checking* (SMC).

From this point and on, we will use M to denote the finite state transition system and $f = \mathbf{AG}\, p$ for a propositional formula p, as the property to be verified.

3.1 Interpolation-Sequence Based Model Checking (ISB)

Note that, an interpolation-sequence exists for a bound N only when the BMC formula φ^N is unsatisfiable, i.e. when there is no counterexample of length N. In case a counterexample exists, BMC returns a counterexample and the interpolation-sequence is not needed.

Definition 3.1.1 (BMC-partitioning)**.** A *BMC-partitioning* for φ^N is the sequence $\Gamma = \langle A_1, A_2, \ldots, A_{N+1} \rangle$ of formulas such that $A_1 = INIT(V^0) \wedge TR(V^0, V^1)$, for every $2 \leq i \leq N$ $A_i = TR(V^{i-1}, V^i)$ and $A_{N+1} = \neg p(V^N)$. Note that $\varphi^N = \bigwedge_{i=1}^{N+1} A_i \ (= \bigwedge \Gamma)$.

For a bound N, consider a BMC formula φ^N and its BMC-partitioning Γ. In case φ^N is unsatisfiable, the interpolation-sequence of Γ is denoted by $\bar{I}^N = \langle I_0^N, I_1^N, \ldots, I_{N+1}^N \rangle$. Note that Γ contains $N+1$ elements and therefore the interpolation-sequence contains $N+2$ elements where the first element and the last one are always $\top$ and $\bot$, respectively.

Next, we intuitively explain our method. We start with $N = 1$. Consider the formula φ^1 and its BMC-partitioning: $\langle A_1, A_2 \rangle$. In case φ^1 is unsatisfiable, there exists an interpolation-sequence of the form $\bar{I}^1 = \langle I_0^1 = \top, I_1^1, I_2^1 = \bot \rangle$. By Definition 2.3.2, $\top \wedge A_1 \Rightarrow I_1^1$ where $A_1 = INIT(V^0) \wedge TR(V^0, V^1)$. Therefore $S_1 \subseteq I_1^1$, where S_1 is the set of states reachable from the initial states in one transition. Also, $I_1^1 \wedge \neg p(V^1)$ is unsatisfiable, since $I_1^1 \wedge A_2 \Rightarrow \bot$, where $A_2 = \neg p(V^1)$. Therefore, $I_1^1 \models p$.

In the next BMC iteration, for $N = 2$, consider φ^2 and its BMC-partitioning $\langle A_1, A_2, A_3 \rangle$. In case φ^2 is unsatisfiable, we get $\bar{I}^2 = \langle \top, I_1^2, I_2^2, \bot \rangle$. Here too, $S_1 \subseteq I_1^2$ and the states reachable from it in one transition are a subset of I_2^2 since $I_1^2 \wedge A_2 \Rightarrow I_2^2$. Also, $S_2 \subseteq I_2^2$ and $I_2^2 \models p$. Let us define the sets $F_1 = I_1^1 \wedge I_1^2$ and $F_2 = I_2^2$. These sets have the following properties, $S_1 \subseteq F_1$, $S_2 \subseteq F_2$, $F_1 \models p$ and $F_2 \models p$. Moreover, $F_1[V^1 \leftarrow V] \wedge TR(V, V') \Rightarrow F_2[V^2 \leftarrow$

$V']$.

In the general case if φ^N is unsatisfiable then for every $1 \leq j \leq N$, $S_j \subseteq I_j^N$. If we now define $F_j = \bigwedge_{k=j}^{N} I_j^k$ then for every $1 \leq j \leq N$ we get:

- $F_j \models p$ since $I_j^j \models p$.
- $F_j \wedge TR(V, V') \Rightarrow F'_{j+1}$ since $I_j^k(V^j) \wedge TR(V^j, V^{j+1}) \Rightarrow I_{j+1}^k(V^{j+1})$ for every $1 \leq k \leq N$
- $S_j \subseteq F_j$ since $S_j \subseteq I_j^k$ for every $1 \leq k \leq N$.

As a result, the sequence $\langle F_0 = INIT, F_1, F_2, \ldots, F_N \rangle$ is a FRS (Definition 2.4.1) and can be used to determine if $M \models f$. Similarly to the sequence obtained from ITP, the sets I_j are over-approximations of S_j computed by SMC. Therefore, these sets can be used to imitate the forward reachability analysis of the model's state-space by means of an over-approximation. This is done in the following manner. BMC runs as usual with one extension. After checking bound N, if a counterexample is found, the algorithm terminates. Otherwise, the interpolation-sequence $\bar{I}^N$ is extracted and the sets F_j for $1 \leq j \leq N$ are updated. If $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$ for some $1 \leq j \leq N$, then we conclude that a fixpoint has been reached and all reachable states have been visited. Thus, $M \models f$. If no fixpoint is found, the bound N is increased and the computation is repeated for $N + 1$. We elaborate mode on fixpoint computation later.

Next, we explain why the algorithm uses $F_j = \bigwedge_{k=j}^{N} I_j^k$ rather than I_j^N in its Nth iteration. Informally, the following facts are needed in order to guarantee the correctness of the algorithm. For every $1 \leq j \leq N$ we need the following:

1. F_j should satisfy p.
2. $F_j(V) \wedge TR(V, V') \Rightarrow F_{j+1}(V')$ for $j \neq N$.

```
1: function UpdateReachable(k,F̄[k],Ī^k)
2:     j = 1
3:     while (j < k) do
4:         F_j = F_j ∧ I_j^k
5:         F̄[k][j] = F_j
6:         j = j + 1
7:     end while
8:     F̄[k][k] = I_k^k
9: end function
```

Figure 3.1: Updating the reachability sequence $\bar{F}_{[k]}$

3. $S_j \subseteq F_j$.

This means that the algorithm cannot be implemented using the extracted interpolation sequence $\bar{I}^N$ alone. This is because $\bar{I}^N$ does not satisfy condition (1): while $I_N^N \models p$, I_j^N for $j \neq N$, does not necessarily satisfy p. This can be remedied by conjoining each I_j^N with I_j^j. However, now condition (2) no longer holds. Taking $F_j = \bigwedge_{k=j}^{N} I_j^k$ results in a sequence with all three properties. By that, the sequence follows the properties of Definition 2.4.1.

The algorithms for updating the FRS and checking for a fixpoint are described in Figure 3.1 and Figure 3.2, respectively. The complete model checking algorithm using the method described above is given in Figure 3.3. We refer to it as *Interpolation-Sequence Based Model Checking* (ISB).

It is important to note that a call to UpdateReachability changes all elements of the FRS $\bar{F}_{[k]}$. Therefore, the function FixpointReached cannot count on inclusion checks done in previous iterations and needs to search for a fixpoint at every point in $\bar{F}_{[k]}$. Moreover, it is not sufficient to check for inclusion of only the last element I_N of $\bar{F}_{[k]}$. Indeed, if there exists $j \leq N$ such that $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$ then all reachable states have been found already. However, the implication $F_N \Rightarrow \bigvee_{i=1}^{N-1} F_i$ might not hold due to additional *unreachable* states in I_N. This is because for all $1 \leq j < N$, F_{j+1} is an over-

```
10: function FixpointReached(F̄[n])
11:     j = 1
12:     while (j ≤ F̄[k].length) do
13:         R = ⋁_{i=0}^{j−1} F_i
14:         φ = F_j ∧ ¬R // Negation of F_j ⇒ R
15:         if (SAT(φ) == false) then return true
16:         end if
17:         j = j + 1
18:     end while
19:     return false
20: end function
```

Figure 3.2: Checking if a fixpoint has been reached

```
21: function ISB(M,f)
22:     k := 0
23:     result = BMC(M, f, 0)
24:     if (result == cex) then
25:         return cex
26:     end if
27:     F̄[k] = ⟨INIT⟩        // Reachability sequence
28:     while (true) do
29:         k = k + 1
30:         result = BMC(M, f, k)
31:         if (result == cex) then
32:             return cex
33:         end if
34:         Ī^k = ⟨⊤, I^k_1, ..., I^k_k, ⊥⟩
35:         UpdateReachable(F̄[k],Ī^k)
36:         if (FixpointReached(F̄[k]) == true) then
37:             return true
38:         end if
39:     end while
40: end function
```

Figure 3.3: The ISB Algorithm

approximation of the states reachable from F_j and not the exact image (that is, $F_j(V) \wedge TR(V,V') \Rightarrow F_{j+1}[V \leftarrow V']$ rather than $F_j(V) \wedge TR(V,V') \equiv F_{j+1}[V \leftarrow V']$).

Theorem 3.1.2. *Assume there is no path of length N or less violating f in M. If there exist $1 < j \leq N$ such that $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$, then $M \models f$.*

Proof. By assumption, there is no path in M of length N or less that violates f. We now show that given $F_j \Rightarrow \bigvee_{i=0}^{j-1} F_i$ we can conclude that there is no path of any length violating f. Let $R = \bigvee_{i=0}^{j-1} F_i$. By assumption, $F_j \Rightarrow R$ and for every $0 \leq i < j$, $F_i(V) \wedge TR(V,V') \Rightarrow F_{i+1}(V')$. Thus, $R(V) \wedge TR(V,V') \Rightarrow R(V')$ (1). Moreover, for every $1 \leq i \leq j$ $F_i \Rightarrow p$. Hence, $R \Rightarrow p$ is unsatisfiable (2).

We can show by induction that all reachable states are in R. The base case handles an initial state. This holds trivially by the definition of R. Now let us assume it holds for all states reachable in k steps. It should be proved for states reachable in $k+1$ steps. Let s_{k+1} be a state reachable in $k+1$ steps from an initial state. Let $\pi = s_0, s_1, \ldots, s_k, s_{k+1}$ be an initial path to s_{k+1}. By the induction hypothesis $s_k \in R$. By the fact that $(s_k, s_{k+1}) \in TR$ and by (1) we can conclude that $s_{k+1} \in R$.

By that and (2), the set of reachable states satisfy p which implies that $M \models f$.

□

Lemma 3.1.3. *Suppose $M \models f$ then there exists a bound N such that $\bar{F} = \{\mathrm{INIT}, F_1, F_2, \ldots, F_N\}$ and there exists an index $1 < j \leq N$ such that $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$.*

Proof. The set of states S is finite. Let us define $N = j = |S| + 1$. $M \models f$ hence for every $0 \leq k \leq N$, φ^k is unsatisfiable. Thus, the interpolation-

sequence $\bar{I}^k$ exists for every $0 \leq k \leq N$ and by that the FRS $\bar{F} = \{INIT, F_1, F_2, \ldots, F_N\}$ exists. Since $|S| < \infty$ we get $F_j \Rightarrow \bigvee_{i=1}^{j-1} F_i$. □

Theorem 3.1.4. *There exists a path π of length N such that π violates f if and only if ISB terminates and returns cex.*

Proof. Assume that the minimal violating path is of length N. For $N-1$ there is no path in M violating f. By Theorem 3.1.2 we get that for every j such that $1 \leq j < N$, $F_j \Rightarrow \bigvee_{i=1}^{j-1} I_i$ does not hold. Therefore, the algorithm cannot terminate by returning *true* in the first $N-1$ iterations. When the algorithm reaches the N-th iteration, $BMC(M, f, N)$ will return a counterexample and the algorithm terminates. The other direction is immediate. □

Theorem 3.1.5. *For every model M and property $f = \mathbf{AG}\, p$ there exists a bound N such that ISB terminates. Moreover,*

- *$M \models f$ if and only if there exists an index $0 < j \leq N$ such that $F_j \Rightarrow \bigvee_{i=0}^{j-1} F_i$.*
- *There exists a path π of length N such that π violates f if and only if ISB returns cex.*

Proof. The proof is immediate from Lemma 3.1.3 and Lemma 3.1.4. □

3.2 Comparing Interpolation-Sequence Based MC to Interpolation Based MC

In the previous section we presented ISB, an algorithm, which combines BMC and interpolation-sequence [57], and in the previous chapter we described the Interpolation based algorithm (ITP) [43]. Both algorithms are based on the

use of interpolation for computing a reachability sequence. In this section we analyze the differences between the algorithms.

Both methods compute an over-approximation of the set of reachable states. However, their state traversals are different. As a result, none is better than the other in all cases. In specific cases, though, one may converge faster.

Several technical details distinguish ISB from ITP. First, the formulas from which the interpolants are extracted are different. For a given bound N, ISB uses the formula φ^N while ITP uses $\varphi^{1,N}$.

Second, the approximated sets are computed in different manners. ISB computes the sets F_j incrementally and refines them after each iteration of BMC, as part of the BMC loop. ITP, on the other hand, recomputes the interpolants whenever the bound is incremented (that is, whenever CHECK-REACHABLE is called with a larger bound).

Third, ISB can be viewed as an addition to the BMC loop. At each application of BMC (with a different bound), the addition includes the extraction of an interpolation-sequence and the check if a fixpoint has been reached. Indeed, after N iterations of the BMC loop in ISB, there are N over-approximated sets of states, $F_1, \ldots, F_N$ satisfying, for each $1 \leq j \leq N$, $S_j \preceq_N F_j$.

On the other hand, ITP consists of two nested loops. The outer loop increments the bounds while the inner loop computes over-approximated sets of reachable states. If the outer loop is at some bound $N > 1$ and the inner loop performs L iterations then there are L sets of states $J_1^N, \ldots, J_L^N$, each satisfying $S_i \preceq_{N+L} J_i^N$ $(1 \leq i \leq L)$. Table 3.1 summarizes the above differences.

In summary, ITP can compute, at a given bound N, as many sets as needed as long as no counterexample is found (not necessarily a real counterexample). On the other hand, for bound N, ISB can only compute N sets. However, it does not need recurrent BMC calls for each bound (only

SMC	ISB	ITP
$\langle S_1, \ldots, S_N \rangle$	$\langle F_1, F_2, \ldots, F_N \rangle$ $S_i \preceq_N F_i$ After checking bounds 1 to N	$\langle J_1^1, J_2^1, \ldots, J_N^1 \rangle$ $S_i \preceq_N J_i^1$ N iterations at bound 1, if possible
$\langle S_1, \ldots, S_{N+L} \rangle$	$\langle F_1, \ldots, F_L, \ldots, F_{N+L} \rangle$ $S_i \preceq_{N+L} F_i$ After checking bounds 1 to $N+L$	$\langle J_1^N, J_2^N, \ldots, J_L^N \rangle$ $S_i \preceq_{N+L} J_i^N$, $(1 \leq i \leq L)$ L iterations at bound N, if possible

Table 3.1: The correlation between the interpolants computed by ISB and ITP to the sets computed by SMC

one is needed). Thus, we can conclude that in cases ITP can compute all the needed sets at a low bound it performs better than ISB. However, for examples where the needed sets can only be computed using higher bounds, ISB has an advantage. This fact is reflected in the experimental results reported in the next section.

As mentioned before, when a counterexample exists the over-approximated sets of reachable states are not needed. If a property is violated then there exists a minimal bound N for which a violating path of length N exists. Both algorithms have to reach this bound in order to find the counterexample. Here, ISB has a clear advantage over ITP. This is because after each BMC run on the original model, ITP executes at least one additional BMC run on a modified model. Thus, ITP invokes at least two BMC runs for each bound from 1 to $N-1$. Clearly, the second BMC run is more demanding than the inclusion check performed by ISB. In all experiments of [57], falsified properties always favored ISB.

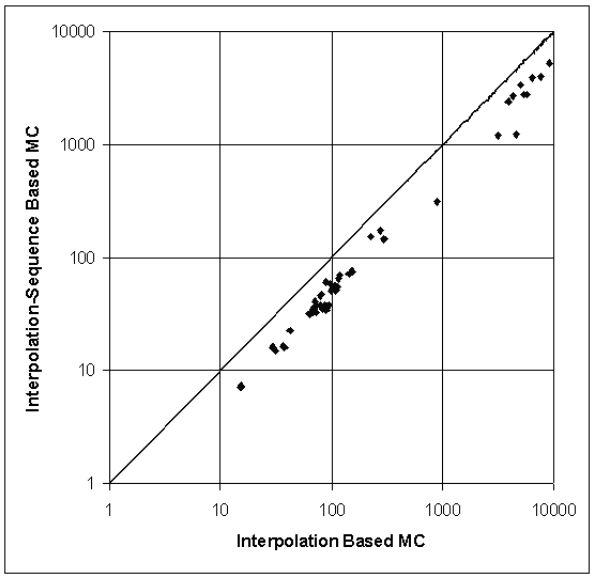


Figure 3.4: Runtime (in seconds) of falsified properties on Intel's micro-architecture.

3.3 Implementation Details and Experimental Results

3.3.1 Implementation Details

Both the ISB and the ITP algorithms were implemented within Intel's verification system using a SAT-based model checker which is based on Intel's in-house SAT solver *Eureka*. The interpolants are represented by a data-structure similar to an And-Inverter Graph (AIG) and are simplified and optimized using known methods such as constant propagation and sharing of redundant expressions.

3.3.2 Experimental Results

The two algorithms have been checked on various models taken from two of Intel's CPU designs. The characteristics of the checked models appear

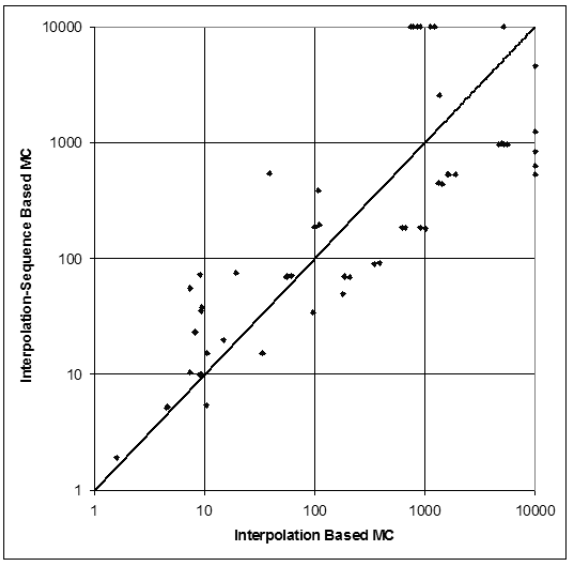


Figure 3.5: Runtime (in seconds) of verified properties on Intel's micro-architecture.

Name	♯Vars	B	B_{ITP}	♯I	♯I_{ITP}	♯BMC	♯BMC$_{ITP}$	Time [s]	Time$_{ITP}$ [s]
f_1	3406	16	15	136	80	16	80	970	5518
f_2	1753	9	8	45	40	9	40	91	388
f_3	1753	7	6	28	28	7	28	49	179
f_4	1753	16	15	136	94	16	94	473	1901
f_5	3406	6	5	21	13	6	13	68	208
f_6	1761	2	1	3	2	2	2	5	4
f_7	3972	3	1	6	3	3	3	19	14
f_8	2197	3	1	6	3	3	3	10	7
f_9	1629	23	6	276	39	23	39	2544	1340
f_{10}	4894	5	1	15	3	5	3	635	101

Table 3.2: Verified properties and their running parameters.
Unindexed columns refer to the ISB algorithm; columns indexed with ITP refer to the ITP algorithm. ♯*Vars* stands for the number of state variables in the cone of influence. B - bound at convergence, ♯I - number of interpolants computed, ♯BMC - number of calls to the BMC algorithm, and *Time[s]* - the runtime in seconds.

in Table 3.3. The 136 properties chosen for the experiments were all real safety properties used to verify the correctness of the designs. The cone of influence for the properties contains thousands of state variables and tens of thousands of gates and signals. The properties vary in that some are *true* and some are *false*. During all checks, a timeout of 10,000 seconds has been set. Experiments were conducted on systems with a dual core Xeon 5160 processors (Core 2 micro-architecture) running at 3.0GHz (4MB L2 cache) with 32GB of main memory. Operating system running on the system is Linux SUSE.

Figure 3.4 and Figure 3.5 show the runtime in seconds for the two algorithms. Each point represents a property from the set of chosen properties. The X axis represents runtime for ITP while the Y axis represents the runtime using ISB. We can see that the results vary. Figure 3.4 shows the runtime for the falsified properties. Figure 3.5 shows the runtime for the verified properties. All falsified properties (total of 67) favor ISB. There are five properties that can be verified by ISB and not by ITP (due to timeout) and two properties that can be falsified using ISB while cannot be falsified using ITP. On the other hand, there are seven properties that cannot be verified by ISB but can be verified by ITP. The rest of the properties (57 total) are all verified by both algorithms.

A more accurate analysis of the algorithms is shown in Table 3.2 that presents running parameters (number of state variables in the cone of influence, bound at convergence, number of interpolants computed, number of calls to BMC and runtime) on various properties for both ITP and ISB. For some cases, even though ITP converges at a lower bound, and computes fewer interpolants than ISB, ISB still converges faster by means of runtime. This is due to the fact that BMC calls are computationally heavier than the extraction of the interpolants. Since ITP issues more calls to BMC than ISB in these cases, the influence on its runtime is noticeable. Through all our experiments, when convergence for ITP could be achieved only at high

Name	♯ Latches	♯ Inputs	♯ Gates
M_1	3611	3	84570
M_2	4968	2079	133255
M_3	12806	402	89392
M_4	1672	459	11195
M_5	19213	305	146717

Table 3.3: Models used for testing

bounds, ISB always performed better while for convergence at lower bounds, ITP performs better. This result is supported by the analysis presented in the previous section.

Chapter 4

Intertwined Forward-Backward Reachability Analysis Using Interpolants

The work we present in this chapter appeared in [59]. We develop a novel SAT-based verification approach which is based on interpolation. The novelty of our approach is in extracting interpolants in both forward and backward manner and exploiting them for an *intertwined approximated forward and backward reachability analysis*. Our approach is also mostly *local* and avoids unrolling of the checked model as much as possible. This results in an efficient and complete SAT-based verification algorithm.

In previous chapters we showed how ITP uses interpolation to extract an over-approximation of a set of reachable states from a proof of unsatisfiability, generated by a SAT-solver. The set of reachable states computed by the reachability analysis is used by ITP to check if a system M satisfies a safety property AGp.

In [8] an alternative SAT-based algorithm, called IC3, is introduced. Similarly to ITP, IC3 also computes over-approximations of sets of reachable states. However, ITP unrolls the model in order to obtain more precise ap-

proximations. In many cases, this is a bottleneck of the approach. IC3, on the other hand, improves the precision of the approximations by performing many local checks that do not require unrolling. We will go into more details about IC3 in Chapter 6.

Both ITP and IC3 compute over-approximations of the sets of states obtained by a *forward reachability analysis*. The forward analysis starts from the initial states of M, and iteratively computes successors while making sure that no bad state violating p is reached. Verification based on reachability can also be performed in a dual manner using a *backward reachability analysis*. The backward analysis starts from the states satisfying $\neg p$ and iteratively computes ancestors while making sure that no initial state is reached.

Traditionally, BDD-based verification methods [20] use both forward and backward analyses [15, 56], while SAT-based methods mainly implement the forward one. Recently, a few works considered backward analysis in the context of SAT as well (e.g. [14, 26]). Most such works use forward and backward analyses independently of each other, or use a weak combination of the two, such as replacing the role of the initial states in the backward analysis by the reachable states computed by a forward analysis.

In the work presented in this chapter, we propose an interpolation-based verification method that applies mostly local checks and avoids unrolling of the model as much as possible. Our approach combines approximated forward and backward analyses in a tight and intertwined way, and uses each of them to enhance the precision of the other. Thus, the tight combination of the two analyses replaces unrolling in enhancing the precision of the computed over-approximated sets of states.

Our work uses the observation that a single SAT check entails information both about states reachable from the initial states (via post-image operations) and about states that reach the bad-states (via pre-image operations). We exemplify this observation by examining the propositional formula $INIT(V) \wedge TR(V, V') \wedge \neg p(V')$ where $INIT$ and $\neg p$ describe the sets of initial

states and bad states, respectively, and $TR(V, V')$ describes the transition relation. If this formula is satisfiable, then there exists a path of length one from the initial states to the bad states. If it is unsatisfiable, then all states reachable from the initial states in one transition are a subset of p. This fact is often used in forward reachability. We now note that the unsatisfiability of this formula can be used in backward reachability as well. This can be done by interpreting it as "all states that can reach the bad states in one transition are disjoint from the initial states".

We exploit this dual observation by extracting two different interpolants from the unsatisfiabe formula $INIT(V) \wedge TR(V, V') \wedge \neg p(V')$. The *forward interpolant* (the one used in ITP) provides an over-approximation of the post-image of *INIT* which is disjoint from $\neg p$. The *backward interpolant*, computed for the same formula when it is read backward, from right to left, provides an over-approximation of the pre-image of $\neg p$ which is disjoint from *INIT*.

We use the above observation as a key element in traversing the state space in a dual fashion, both forward from the initial states and backwards from the bad states.

Our algorithm, *Dual Approximated Reachability* (DAR), computes a *Forward Reachability Sequence* $\bar{F} = \langle F_0, F_1, \ldots \rangle$, and a *Backward Reachability Sequence* $\bar{B} = \langle B_0, B_1, \ldots \rangle$. The set F_i represents an over-approximation of the set of states which are reachable from *INIT* in exactly i transitions. Further, F_i is disjoint from $\neg p$. Similarly, B_i represents an over-approximation of the set of states that can reach $\neg p$ in exactly i transitions, and it is also disjoint from *INIT*. Thus, the existence of either $\bar{F}$ or $\bar{B}$ of length n ensures that no counterexample of length n exists in M.

The goal of DAR is to gradually strengthen (make more precise) and extend $\bar{F}$ and $\bar{B}$, until a counterexample is found or until one of $\bar{F}$ or $\bar{B}$ reaches a *fixpoint*, that is, no new states are found when the sequence is further extended. To do this, DAR employs local strengthening phases, assisted by

a global strengthening phase, when needed. Only the global strengthening involves unrolling. Thus, the number of unrolling applications is limited. In addition, the depth of the unrolling is also limited.

Initially, $\bar{F} = \langle F_0 \rangle$ and $\bar{B} = \langle B_0 \rangle$, where $F_0 = \mathit{INIT}$ and $B_0 = \neg p$. At iteration n, we define the sequence $\Pi = \langle\ \mathit{INIT},\ F_1 \wedge B_n,\ F_2 \wedge B_{n-1},\ \ldots,\ F_n \wedge B_1,\ \neg p\ \rangle$. Π represents an over-approximation of the set of all possible paths from INIT to $\neg p$ of length $n+1$ in M. That is, Π over-approximates the set of all counterexamples in M of length $n+1$. DAR attempts to show that Π represents no counterexample.

The *local strengthening phase* checks whether there are in fact transitions between every two consecutive sets in Π. It turns out that this can be done by applying local checks of the form $F_i(V) \wedge \mathit{TR}(V, V') \wedge B_{n-i}(V')$. If such a formula is unsatisfiable, then no transition exists from $F_i \wedge B_{n-i+1}$ to its successor along Π, thus no counterexample of length $n+1$ exists. This can also be understood by observing that the unsatisfiability of $F_i(V) \wedge \mathit{TR}(V, V') \wedge B_{n-i}(V')$ means that the states reachable from the initial states in i transitions cannot reach B_{n-i} in one transition. Since B_{n-i} includes all states reaching $\neg p$ in $n-i$ transitions, no counterexample of length $n+1$ exists.

In this case, the forward interpolant of $F_i(V) \wedge \mathit{TR}(V, V') \wedge B_{n-i}(V')$ is used to strengthen F_{i+1} while the backward interpolant strengthens B_{n-i+1}. Strengthening is now propagated along $\bar{F}$ and $\bar{B}$. This reflects the fact that the components of one sequence are strengthened based on the components of the other everywhere along the sequences, making the analyses closely intertwined. Next, $\bar{F}$ and $\bar{B}$ are extended by initializing F_{n+1} to be the forward interpolant of $F_n(V) \wedge \mathit{TR}(V, V') \wedge B_0(V')$ and B_{n+1} to be the backward interpolant of $F_0(V) \wedge \mathit{TR}(V, V') \wedge B_n(V')$.

The *global strengthening phase* is applied when $F_i(V) \wedge \mathit{TR}(V, V') \wedge B_{n-i}(V')$ is satisfiable for *all* i. This implies that a transition exists between every two consecutive sets in Π, making local reasoning insufficient. We therefore grad-

ually unroll the model M and check whether the states in $F_i \wedge B_{n-i+1}$ are *unreachable* from *INIT* via i transitions of M. Once we find such an i, the unrolling can stop. We are certain that no counterexample of length $n+1$ exists. We strengthen $\bar{F}$ up to depth i using an interpolation-sequence [38], and return to the local strengthening phase for further strengthening and for extending $\bar{F}$ and $\bar{B}$ to length $n+1$.

We implemented our DAR algorithm and compared it to both ITP and IC3, on real-life industrial designs as well as examples from the HWMCC'11 benchmark. In many cases, our algorithm outperformed both methods. We noticed that the number of iterations where global strengthening was needed, as well as the depth of the unrolling in the global strengthening phase is often smaller relative to the length of $\bar{F}$ and $\bar{B}$. This reflects the fact that our use of unrolling is limited.

To summarize, the novelty of our approach is twofold. It suggests a SAT-based intertwined forward-backward reachability analysis. Further, the reachability analysis is interpolation-based. Yet, it is mostly local and avoids unrolling as much as possible.

4.1 Related Work

Several works use interpolation in the context of model checking. Interpolation-based model checking (ITP) was initially introduced in [43]. Similarly to ITP, DAR also uses interpolation to compute over-approximated sets of reachable states. However, ITP computes interpolants based on an unrolled formula and increases unrolling to make the over-approximation more precise. DAR, on the other hand, mostly avoids unrolling and uses backward and forward interpolants from local checks for strengthening. In addition, ITP restarts when it finds a spurious counterexample, increasing the depth of unrolling. In contrast, DAR keeps strengthening the computed over-approximations from previous iterations. In [14] improvements for ITP are suggested. They im-

plement a backward-traversal using interpolants. Unlike our method, their backward traversal is an adaptation of ITP and is not tightly integrated with the forward traversal.

The work in [26] is also based on ITP in the sense of computing interpolants based on unrolling of the model, where the depth of unrolling increases in each iteration. Their work integrates the use of forward and backward analyses: in each iteration the result of the backward analysis is used to restrict the initial states and the result of the forward analysis is used to restrict the bad states. Our approach, on the other hand, uses the result of the forward analysis to strengthen *all intermediate sets* of $\bar{B}$. Similarly the result of the backward analysis stregthens $\bar{F}$.

Interpolation-sequence, which extends the notion of an interpolant for a sequence of formulas has been proposed and used for model checking [38, 44, 57, 16]. DAR makes a similar use of interpolation-sequence in its global strengthening phase. In contrast to the other methods, interpolation-sequence is not a key element of DAR since it is only applied occasionally. Further, it is applied to a restricted depth of unrolling.

The introduction of IC3 [8] suggested a different way to compute information about reachable states. During this process, sets of states that are similar in characteristics to interpolants are computed. Unlike interpolation-based approaches IC3 requires no unrolling and is based on inductive reasoning. The main difference between DAR and IC3 is in the way they strengthen the over-approximated sets of states. IC3 finds a state that can reach $\neg p$ and if it concludes that this state is not reachable, it tries to generalize this fact and removes more than just one state. DAR on the other hand finds an over-approximation of *all* states that can reach $\neg p$, rather than a single state. It then tries to prove that the entire set is unreachable. Also, when DAR fails to strengthen using local reasoning, it applies a limited unrolling in the global phase. On the other hand, IC3 can fall into state enumeration if generalization is not successful.

4.2 Using Interpolants for Forward and Backward Analysis

Let Q be a propositional formula over V. The *post-image* of Q w.r.t. M is the set of all states reachable from Q in one transition, defined by the formula $\exists V[Q(V) \wedge TR(V, V')]$ (note that this formula is defined over V'). The *pre-image* of Q w.r.t. M is the set of all states that can reach Q in one transition, defined by $\exists V'[TR(V, V') \wedge Q(V')]$.

Definition 4.2.1. Let M be a transition system and let φ and ψ be propositional formulas over V. We define the formula $\Gamma_M(\varphi, \psi) = \varphi(V) \wedge TR(V, V') \wedge \psi(V')$ to be a *local reachability check* w.r.t. M, φ and ψ.

Whenever M is clear from the context we omit M and write $\Gamma(\varphi, \psi)$.

4.2.1 Forward and Backward Interpolants

Let R and Q be propositional formulas over V representing sets of states, and let $TR(V, V')$ be a transition relation. Suppose we would like to know if the post image of R, i.e., the set of states reachable from R in one transition, is disjoint from Q. This property can be checked by checking the formula $\Gamma(R, Q) = R(V) \wedge TR(V, V') \wedge Q(V')$ for unsatisfiability. If the formula is unsatisfiable then the answer is yes, meaning that Q is not reachable from R in one step. Moreover, using interpolation enables us to derive from the unsatisfiable formula an over-approximation of the post image of R that is still disjoint from Q. Specifically, let $\varphi^- = R(V) \wedge TR(V, V')$ and $\varphi^+ = Q(V')$. An interpolant $I = I(\varphi^-, \varphi^+)$ satisfies $R(V) \wedge TR(V, V') \Rightarrow I(V')$ and $I(V') \wedge Q(V') \equiv \bot$. Therefore, I represents an over approximation of the states reachable from R in one transition, and it is also disjoint from Q.

The unsatisfiability of the formula $\Gamma(R, Q) = R(V) \wedge TR(V, V') \wedge Q(V')$ can also be interpreted in a different manner, shedding light on the pre-image of Q. More precisely, the unsatisfiability of the formula states that

the pre-image of Q, i.e., the set of all states that can reach Q in one transition, is disjoint from R. This view leads to a different way of using interpolation in this setting. For the backward interpretation, we now define $\varphi^- = TR(V, V') \wedge Q(V')$ and $\varphi^+ = R(V)$. Again, since $\varphi^- \wedge \varphi^+$ is unsatisfiable, an interpolant I exists. Formally $TR(V, V') \wedge Q(V') \Rightarrow I(V)$, therefore I is an over-approximation of the pre-image of Q. Moreover, $I \wedge R$ is unsatisfiable and therefore I is disjoint from R.

We conclude that interpolation gives us a way to approximate both post-image and pre-image computations. Formally, we define forward and backward interpolants:

Definition 4.2.2 (Forward and Backward Interpolants)**.** Let R and Q be propositional formulas over V s.t. $\Gamma(R, Q) \equiv \bot$. The *forward interpolant* of $\Gamma(R, Q)$, denoted $\mathrm{FI}(R, Q)$, is $I(R(V) \wedge TR(V, V'), Q(V'))[V' \leftarrow V]$. The *backward interpolant* of $\Gamma(R, Q)$, denoted $\mathrm{BI}(R, Q)$, is $I(TR(V, V') \wedge Q(V'), R(V))$.

Note that $I(R(V) \wedge TR(V, V'), Q(V'))$ is defined over V'. Therefore, we substitute V for V' in the definition of a forward interpolant. As explained above:

Theorem 4.2.3. *$FI(R, Q)$ over-approximates the post-image of R, and is disjoint from Q. Similarly, $BI(R, Q)$ over-approximates the pre-image of Q, and is disjoint from R.*

Proof. By definition, $\mathrm{FI}(R, Q) = I(R(V) \wedge TR(V, V'), Q(V'))[V' \leftarrow V]$. By the properties of an interpolant, $R(V) \wedge TR(V, V') \Rightarrow I(R(V) \wedge TR(V, V'), Q(V'))$. Recall that $I(R(V) \wedge TR(V, V'), Q(V'))$ is defined over V' only. Therefore, the above implication implies that $\exists V[R(V) \wedge TR(V, V')] \Rightarrow I(R(V) \wedge TR(V, V'), Q(V'))$ which ensures that the interpolant over-approximates the post-image of R. In addition, $I(R(V) \wedge TR(V, V'), Q(V')) \wedge Q(V') \equiv \bot$, which ensures that the interpolant is disjoint from Q. The proof for $\mathrm{BI}(R, Q)$ is similar. □

4.2.2 Forward and Backward Reachability Sequences

Our model checking algorithm for safety properties, described in Section 4.3, uses forward and backward interpolants for the computation of over-approximated sets of forward and backward reachable states. Technically, we consider both forward and backward reachability approximations:

Recall the definition of a FRS (Definition 2.4.1). We now define the dual backward reachability sequence.

Definition 4.2.4. A *Backward Reachability Sequence (BRS)* w.r.t. M and a property AGp is a sequence $\bar{B}_{[n]} = \langle B_0, B_1, \ldots, B_n \rangle$ of sets of states s.t.

- $B_0 = \neg p$.
- $B_{i+1}(V) \Leftarrow TR(V, V') \wedge B_i(V')$ for $0 < i \leq n$.
- $B_i \Rightarrow \neg INIT$ for $0 \leq i \leq n$.

We define the *length* of $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ to be n. When n is clear from the context, we omit it from the notation and simply use $\bar{F}$ and $\bar{B}$. The second condition in Definition 2.4.1 (Definition 4.2.4) states that F_{i+1} (B_{i+1}) is an over-approximation of the post(pre)-image of F_i (B_i) w.r.t. M. Iterating these properties, we conclude that F_i over-approximates the set of states reachable from $INIT$ in i steps, and B_i over-approximates the set of states reaching a violation of p in i steps. The following properties hold for FRS and BRS:

Theorem 4.2.5. *A FRS of length n for M and AGp exists iff there is no counterexample of length n or less in M for AGp. Similarly for a BRS.*

Proof. Consider a FRS of length n. Denote by R_i the set of states reachable from $INIT$ in exactly i steps. It can be shown inductively that $R_i \Rightarrow F_i$: Initially, $F_0 = INIT = R_0$, and for the induction step: $R_i(V) \wedge TR(V, V') \Rightarrow F_i(V) \wedge TR(V, V') \Rightarrow F_{i+1}(V')$. Since the right-hand side is defined over

V' only, this implication implies that $R_{i+1}(V') = \exists V[R_i(V) \wedge TR(V,V')] \Rightarrow F_{i+1}(V')$.

Now suppose there exists a FRS of length n. For every $0 \leq i \leq n$, $R_i \Rightarrow F_i \Rightarrow p$ (the second implication is due to the third property of a FRS). This means that all states reachable from *INIT* in n steps or less satisfy p. Hence no counterexample of length n or less exists.

For the other direction, suppose there is no counterexample of length n or less. Then a FRS of length n can be defined by setting $F_i = R_i$ for every $0 \leq i \leq n$. The sequence $\langle R_0, \ldots, R_n \rangle$ satisfies all the requirement of a FRS: $R_0 = INIT$. $R_{i+1}(V') = \exists V[R_i(V) \wedge TR(V,V')]$. Therefore, $R_{i+1}(V') \Leftarrow R_i(V) \wedge TR(V,V')$. Finally, since no counterexample of length $\leq n$ exists, $R_i \Rightarrow p$ for every $0 \leq i \leq n$.

The proof for a BRS is similar, when replacing R_i with the set Q_i of states that reach $\neg p$ in exactly i steps.

□

Next we extend the definition of a *fixpoint* (Definition 2.4.2) and the resulting theorem (Theorem 2.4.3) to include also the BRS:

Definition 4.2.6 (Fixpoint)**.** Let $\bar{F}$ be a FRS and $\bar{B}$ a BRS of length n. We say that $\bar{F}$ is at *fixpoint* if there exists $0 < k \leq n$ s.t. $F_k \Rightarrow \bigvee_{i=0}^{k-1} F_i$. Similarly, we say that $\bar{B}$ is at *fixpoint* if there exists $0 < k \leq n$ s.t. $B_k \Rightarrow \bigvee_{i=0}^{k-1} B_i$.

Theorem 4.2.7. *Let $\bar{F}$ be a FRS and $\bar{B}$ a BRS of length n for M and AGp. If $\bar{F}$ or $\bar{B}$ is at fixpoint then $M \models AGp$.*

Proof. The proof for $\bar{F}$ is similar to the one appearing for Theorem 2.4.3.

The proof for $\bar{B}$ is similar, where instead of R we use the set Q of all states that can reach $\neg p$ in any number of steps. □ □

Note that a fixpoint in one of the sequences suffices to conclude that $M \models AGp$.

4.3 Dual Approximated Reachability

In this section we describe our Dual Approximated Reachability (DAR) algorithm for model checking safety properties. DAR computes over-approximated sets of reachable states for both forward and backward reachability analysis by means of a FRS and a BRS, using interpolants. The computations are intertwined where each of them is used to make the other tighter. DAR avoids unrolling of the transition system unless it is really needed.

Technically, DAR computes a FRS $\bar{F}$ and a BRS $\bar{B}$ and gradually extends them until either a counterexample is found or a fixpoint is reached on either $\bar{F}$ or $\bar{B}$. Since the state-space of M is finite, one of the above is bound to happen, which ensures that:

Theorem 4.3.1. *Given a model M and a safety property $\varphi = AGp$, DAR always terminates. Moreover, $M \models \varphi$ if and only if DAR returns "Verified".*

We defer the proof of Theorem. 4.3.1 to the end of the section. We now describe DAR in detail. The pseudocode of DAR appears in Figure 4.1.

Initialization of DAR (lines 2-5) starts by checking the formula $INIT \wedge \neg p$. If this formula is unsatisfiable, the initial states of M satisfy the property. If not, a counterexample exists. In the former case, DAR initializes $\bar{F} = \langle F_0 = INIT \rangle$ and $\bar{B} = \langle B_0 = \neg p \rangle$. Clearly $\bar{F}$ and $\bar{B}$ are FRS and BRS, respectively.

The iterative part of DAR (lines 8-13) then gradually extends and strengthens $\bar{F}$ and $\bar{B}$ s.t. they remain a FRS and a BRS respectively. As ensured by Lemma 4.2.5, this is possible as long as no counterexample of the corresponding length exists. In the following, we describe a single iteration of DAR, strengthening and extending $\bar{F}$ and $\bar{B}$, or reporting a counterexample.

4.3.1 First Iteration of DAR

Let us first present the first iteration of DAR. Recall that initially $\bar{F} = \langle F_0 = INIT \rangle$ and $\bar{B} = \langle B_0 = \neg p \rangle$. DAR then checks the formula $F_0 \wedge TR \wedge B'_0 =$

```
1: function DAR(M,p)
2:     if INIT ∧ ¬p == SAT then
3:         return cex
4:     end if
5:     F̄ = ⟨F_0 = INIT⟩, B̄ = ⟨B_0 = ¬p⟩
6:     n = 0
7:     while !F̄.FIXPOINT()∧!B̄.FIXPOINT() do
8:         if LOCSTRENGTHEN(F̄, B̄, n) == false then
9:             if GLBSTRENGTHEN(F̄, B̄, n) == false then
10:                return cex
11:            end if
12:        end if
13:        n = n + 1
14:    end while
15:    return Verified
16: end function
```

Figure 4.1: Dual Approximated Reachability

$INIT \wedge TR \wedge \neg p'$ for satisfiability. In case this formula is satisfiable a counterexample of length one exists. Otherwise, the formula is unsatisfiable, meaning all states reachable from $INIT$ in one transition satisfy p. Alternatively, all states that can reach $\neg p$ in one transition are not part of $INIT$. Thus, the unsatisfiability of $INIT \wedge TR \wedge \neg p'$ entails information both about the post-image of $INIT$ and about the pre-image of $\neg p$.

The above gives us an intuition of how to extend both the FRS and the BRS. For the FRS we define $F_1 = \mathrm{FI}(F_0, B_0)$. For the extension of the BRS, we define $B_1 = \mathrm{BI}(F_0, B_0)$. Due to the properties of the interpolants, the sequences $\bar{F} = \langle F_0, F_1 \rangle$ and $\bar{B} = \langle B_0, B_1 \rangle$ are a FRS and a BRS respectively.

4.3.2 General Iteration of DAR

Let us now discuss a general iteration $n + 1$. Consider the FRS $\bar{F}_{[n]} = \langle F_0, F_1, \ldots, F_n \rangle$ and the BRS $\bar{B}_{[n]} = \langle B_0, B_1, \ldots B_n \rangle$ obtained at iteration n. The goal of iteration $n + 1$ is to check if a counterexample of length $n + 1$

exists, and if not, extend these sequences to length $n+1$ s.t. they remain a FRS and a BRS.

The combination of $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ provides an approximate description of all possible counterexamples of length $n+1$ in M. Namely, recall that F_i over-approximates the set of all states reachable from *INIT* in i steps. Similarly, B_j over-approximates the set of all states that can reach $\neg p$ in j steps. Their intersection, $F_i \wedge B_j$ therefore over-approximates the set of all states that are both reachable from *INIT* in i steps and can reach $\neg p$ in j steps. These are states that appear in the i-th step of a counterexample of length $i+j$ for AGp in M. In particular, when we align $\bar{F}$ and $\bar{B}$ one against the other, conjoining F_i with B_{n-i+1}, we obtain an over-approximation of the set of all states that appear in the i-th step of a counterexample of length $n+1$ for AGp in M. The sequence

$$\Pi(\bar{F}_{[n]}, \bar{B}_{[n]}) = \langle \mathit{INIT}, F_1 \wedge B_n, F_2 \wedge B_{n-1}, \ldots, F_n \wedge B_1, \neg p \rangle$$

therefore over-approximates the set of *all* counterexamples of length $n+1$ in M for AGp.

We refer to the sequence $\Pi(\bar{F}_{[n]}, \bar{B}_{[n]})$ as an *approximated Counterexample* (aCEX). Whenever clear from the context we write Π and refer to the i-th element in the sequence as Π_i. A sequence of states $s_0, \ldots, s_{n+1}$ in M *matches* Π if for every $0 \leq i \leq n+1$, $s_i \in \Pi_i$. Formally, Π has the following property.

Lemma 4.3.2. *Let $\pi = s_0, \ldots, s_{n+1}$ be a counterexample in M. Then, π matches Π.*

Proof. Since π is a path in M, for every $0 \leq i \leq n$, we have that s_i is reachable in i steps from *INIT*. Therefore, $s_i \in F_i$. In addition, since π is a counterexample, $s_{n+1} \models \neg p$. This ensures that for every $1 \leq i \leq n+1$, s_i can reach $\neg p$ in $n-i+1$ transitions. Therefore, $s_i \in B_{n-i+1}$. We conclude that $s_0 \in F_0 = \Pi_0$, $s_{n+1} \in B_0 = \Pi_{n+1}$, and for every $1 \leq i \leq n$, $s_i \in F_i \wedge B_{n-i+1} = \Pi_i$. Therefore π matches Π. □ □

By Lemma 4.3.2, checking if a counterexample exists amounts to checking if some path matches Π. Such a path is necessarily a counterexample of length $n+1$. If such a path exists, we say that Π is *valid*.

DAR first attempts to check for (in)validity of the aCEX using local checks in a *local strengthening phase*. If this fails, DAR moves on to the *global strengthening phase* that applies global checks. In both phases, if the invalidity of the aCEX is established, the FRS and BRS are strengthened and extended into a FRS and a BRS of length $n+1$. Otherwise, the aCEX is found to be valid and a counterexample of length $n+1$ is obtained in the process.

4.3.2.1 Local Strengthening Phase

The local strengthening phase aims at checking if Π is *locally invalid*, which provides a sufficient condition for its invalidity.

Theorem 4.3.3. Π *is* locally invalid *if there exists* $0 \leq i \leq n$ *s.t.* $\Gamma(\Pi_i, \Pi_{i+1}) \equiv \bot$.

Theorem 4.3.4. *If* Π *is locally invalid, then it is also invalid.*

Proof. Suppose $\Gamma(\Pi_i, \Pi_{i+1}) = \Pi_i \wedge TR \wedge \Pi'_{i+1} \equiv \bot$ for some $0 \leq i \leq$. Assume to the contrary that a counterexample $s_0, \ldots, s_{n+1}$ exists in M. By Lemma 4.3.2, it matches Π. In particular, $s_i \in \Pi_i$ and $s_{i+1} \in \Pi_{i+1}$. Moreover, since $s_0, \ldots, s_{n+1}$ is a path in M, $(s_i, s_{i+1}) \in TR$, which implies that $\Pi_i \wedge TR \wedge \Pi'_{i+1} \not\equiv \bot$, in contradiction. □ □

In order to check if Π is locally invalid, we use the following observation.

Lemma 4.3.5. *Let* $\bar{F}_{[n]}$ *be a FRS,* $\bar{B}_{[n]}$ *be a BRS, and* $1 \leq i \leq n$. *Then* $\Gamma(F_i \wedge B_{n-i+1}, F_{i+1} \wedge B_{n-i}) \equiv \Gamma(F_i, B_{n-i})$. *Similarly,* $\Gamma(\mathrm{INIT}, F_1 \wedge B_n) \equiv \Gamma(F_0, B_n)$, *and* $\Gamma(F_n \wedge B_1, \neg p) \equiv \Gamma(F_n, B_0)$. *We conclude that for every* $0 \leq i \leq n$, $\Gamma(\Pi_i, \Pi_{i+1}) \equiv \bot$ *iff* $\Gamma(F_i, B_{n-i}) \equiv \bot$.

Proof. We first show that $\Gamma(F_i \wedge B_{n-i+1}, F_{i+1} \wedge B_{n-i}) \equiv \Gamma(F_i, B_{n-i})$. This follows from the property of a FRS, where $F_i(V) \wedge TR(V, V') \Rightarrow F_{i+1}(V')$, and the property of a BRS, where $B_{n-i+1}(V) \Leftarrow TR(V, V') \wedge B_{n-i}(V')$. Specifically, since $F_i(V) \wedge TR(V, V') \Rightarrow F_{i+1}(V')$, we get that $F_i(V) \wedge TR(V, V') \wedge B_{n-i}(V') \Rightarrow F_{i+1}(V')$, and as a result $\Gamma(F_i, B_{n-i}) = F_i(V) \wedge TR(V, V') \wedge B_{n-i}(V') \equiv F_i(V) \wedge TR(V, V') \wedge B_{n-i}(V') \wedge F_{i+1}(V')$ (1). Similarly, since $B_{n-i+1}(V) \Leftarrow TR(V, V') \wedge B_{n-i}(V')$, we conclude that $B_{n-i+1}(V) \Leftarrow F_i(V) \wedge TR(V, V') \wedge B_{n-i}(V') \wedge F_{i+1}(V')$ holds as well, which means that $F_i(V) \wedge TR(V, V') \wedge B_{n-i}(V') \wedge F_{i+1}(V') \equiv F_i(V) \wedge B_{n-i+1}(V) \wedge TR(V, V') \wedge B_{n-i}(V') \wedge F_{i+1}(V') = \Gamma(F_i \wedge B_{n-i+1}, F_{i+1} \wedge B_{n-i})$ (2). Finally, by concatenating the sequence of equivalences in (1) and (2) we conclude that $\Gamma(F_i, B_{n-i}) \equiv \Gamma(F_i \wedge B_{n-i+1}, F_{i+1} \wedge B_{n-i})$.

The proof for $\Gamma(\mathit{INIT}, F_1 \wedge B_n) \equiv \Gamma(F_0, B_n)$ is similar, except that only $F_0(V) \wedge TR(V, V') \Rightarrow F_1(V')$ is used. Specifically, because of the above, $F_0(V) \wedge TR(V, V') \wedge B_n(V') \Rightarrow F_1(V')$. Therefore, $\Gamma(F_0, B_n) = F_0(V) \wedge TR(V, V') \wedge B_n(V') \equiv F_0(V) \wedge TR(V, V') \wedge B_n(V') \wedge F_1(V') = \Gamma(F_0, F_1 \wedge B_n) = \Gamma(\mathit{INIT}, F_1 \wedge B_n)$. Dually, in the proof of $\Gamma(F_n \wedge B_1, \neg p) \equiv \Gamma(F_n, B_0)$, only $B_1(V) \Leftarrow TR(V, V') \wedge B_0(V')$ is used.

The conclusion described in the second part of the lemma results from the fact that $\Gamma(\Pi_0, \Pi_1) = \Gamma(\mathit{INIT}, F_1 \wedge B_n)$, $\Gamma(\Pi_n, \Pi_{n+1}) = \Gamma(F_n \wedge B_1, \neg p)$ and for every $1 \leq i \leq n$, $\Gamma(\Pi_i, \Pi_{i+1}) = \Gamma(F_i \wedge B_{n-i+1}, F_{i+1} \wedge B_{n-i})$. □ □

Lemma 4.3.5 implies that if there exists $0 \leq i \leq n$ s.t. $\Gamma(F_i, B_{n-i}) \equiv \bot$, then the aCEX is locally invalid and hence invalid. This can also be understood intuitively, as the above means that the (over-approximated) set of states reachable from *INIT* in i steps and the (over-approximated) set of states that can reach $\neg p$ in $n - i$ steps are not reachable from one another in one step. This means that altogether $\neg p$ is not reachable from *INIT* in $i + (n - i) + 1 = n + 1$ steps, and hence no counterexample of length $n + 1$ exists.

In the local strengthening phase, DAR therefore searches for an index

$0 \leq i \leq n$ s.t. $\Gamma(F_i, B_{n-i}) \equiv \bot$. It starts by checking the formula $\Gamma(F_n, B_0)$, setting $i = n$. In case it is satisfiable, DAR starts to iteratively go backwards along $\bar{F}$ and $\bar{B}$ decreasing i by 1. The traversal continues until either $\Gamma(F_i, B_{n-i})$ turns out to be unsatisfiable for some $0 \leq i \leq n$ or until $\Gamma(F_0, B_n)$ is found to be satisfiable.

If an index i is found s.t. $\Gamma(F_i, B_{n-i}) \equiv \bot$, then the aCEX is locally invalid and by Lemma 4.3.4 we conclude that no counterexample of length $n + 1$ exists. Moreover, in this case, the FRS and BRS are locally and gradually strengthened and extended as follows.

Iterative Local Strengthening: Iterative local strengthening is reached when it is already known that no counterexample of length $n+1$ exists. Thus, as Lemma 4.2.5 ensures, there exist a FRS and BRS of length $n+1$. However, $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ cannot necessarily be extended immediately. For example, if $\Gamma(F_n, B_0) = F_n(V) \wedge TR(V, V') \wedge \neg p(V') \not\equiv \bot$, then no F_{n+1} can be obtained s.t. $F_n(V) \wedge TR(V, V') \Rightarrow F_{n+1}(V')$ and in addition $F_{n+1} \Rightarrow p$. On the other hand, if $\Gamma(F_n, B_0) \equiv \bot$ then F_{n+1} can be initialized using $\mathrm{FI}(F_n, B_0)$ while maintaining the properties of a FRS (similarly to the initialization of F_1). Dually, if $\Gamma(F_0, B_n) \not\equiv \bot$, then no extension of $\bar{B}_{[n]}$ is possible, while if $\Gamma(F_0, B_n) \equiv \bot$, we can set $B_{n+1} = \mathrm{BI}(F_0, B_n)$. We therefore first strengthen the components of $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ until $\Gamma(F_n, B_0) \equiv \bot$ and $\Gamma(F_0, B_n) \equiv \bot$, which is a necessary and sufficient condition for extending $\bar{F}$ and $\bar{B}$.

Recall that $\Gamma(F_i, B_{n-i}) \equiv \bot$ for some $0 \leq i \leq n$. This means that even though the components of $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ may not be precise enough to enable their extension, they are precise enough at least in one place that allowed us to conclude that no counterexample of length $n + 1$ exists. DAR uses this "local" precision to strengthen the entire sequences, as described below.

In order to simplify the references to the indices, we replace the use of i and $n - i$ by $0 \leq i, j \leq n$ s.t. $i + j = n$. Therefore $\Gamma(F_i, B_j) \equiv \bot$ for some $0 \leq i, j \leq n$ s.t. $i + j = n$. This ensures that there exists a forward

interpolant $\mathrm{FI}(F_i, B_j)$, as well as a backward interpolant $\mathrm{BI}(F_i, B_j)$. We can therefore perform a *local strengthening step* updating F_{i+1} and B_{j+1}:

Definition 4.3.6. Let $\bar{F}_{[n]}$ be a FRS and $\bar{B}_{[n]}$ be a BRS s.t. $\Gamma(F_i, B_j) \equiv \bot$ for some $0 \leq i, j \leq n$ s.t. $i + j = n$. A *forward strengthening step at* (i, j) strengthens $\bar{F}_{[n]}$: If $i < n$, $F_{i+1} = F_{i+1} \wedge \mathrm{FI}(F_i, B_j)$. A *backward strengthening step at* (i, j) strengthens $\bar{B}_{[n]}$: If $j < n$, $B_{j+1} = B_{j+1} \wedge \mathrm{BI}(F_i, B_j)$.

We refer to $i, j < n$ since F_{n+1} and B_{n+1} are not yet defined and therefore cannot be updated. The strengthening propagates the unsatisfiability of $\Gamma(F_i, B_j)$ one step forward and one step backward:

Lemma 4.3.7. *Let $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ be the result of a forward or backward strengthening step at (i, j) s.t. $i + j = n$. Then*

- *For a forward strengthening step, if $i < n$, $\Gamma(F_{i+1}, B_{j-1}) \equiv \bot$.*
- *For a backward strengthening step, if $j < n$, $\Gamma(F_{i-1}, B_{j+1}) \equiv \bot$.*

Proof. We first show that $\bar{F}_{[n]}$ remains a FRS. The proof for $\bar{B}_{[n]}$ is similar. Recall that $\bar{F}_{[n]}$ was updated by setting $F_{i+1} = F_{i+1} \wedge \mathrm{FI}(F_i, B_j)$. The other components have not changed. It therefore suffices to show that after this update the following properties still hold: (1) $F_{i+1} \Rightarrow p$, (2) $F_i(V) \wedge TR(V, V') \Rightarrow F_{i+1}(V')$, and (3) $F_{i+1}(V) \wedge TR(V, V') \Rightarrow F_{i+2}(V')$. For the sake of the proof, we denote by F^o_{i+1} the old copy of F_{i+1}, before the update. We therefore have that $F_{i+1} = F^o_{i+1} \wedge \mathrm{FI}(F_i, B_j)$, and in particular $F_{i+1} \Rightarrow F^o_{i+1}$.

1. $F_{i+1} \Rightarrow p$: holds since $F_{i+1} \Rightarrow F^o_{i+1} \Rightarrow p$.
2. $F_i(V) \wedge TR(V, V') \Rightarrow F_{i+1}(V')$: holds since $F_i(V) \wedge TR(V, V') \Rightarrow F^o_{i+1}(V')$. In addition, by the properties of an interpolant, $F_i(V) \wedge TR(V, V') \Rightarrow \mathrm{FI}(F_i, B_j)(V')$. Conjoining the two implications, we conclude that $F_i(V) \wedge TR(V, V') \Rightarrow F^o_{i+1}(V') \wedge \mathrm{FI}(F_i, B_j)(V') = F_{i+1}(V')$.

3. $F_{i+1}(V) \wedge TR(V,V') \Rightarrow F_{i+2}(V')$: holds since $F_{i+1} \Rightarrow F^o_{i+1}$, which implies that $F_{i+1}(V) \wedge TR(V,V') \Rightarrow F^o_{i+1}(V) \wedge TR(V,V') \Rightarrow F_{i+2}(V')$.

We now consider the second part of the lemma. Consider first a forward strengthening step for $i < n$ (and accordingly $j > 0$), where $F_{i+1} = F_{i+1} \wedge \mathrm{FI}(F_i, B_j)$. By the properties of an interpolant, we know that $\mathrm{FI}(F_i, B_j) \wedge B_j \equiv \bot$. Therefore, after the update, $F_{i+1} \wedge B_j \equiv \bot$. This implies that $\Gamma(F_{i+1}, B_{j-1}) \equiv \bot$: Recall that $B_j(V) \Leftarrow TR(V,V') \wedge B_{j-1}(V')$. Therefore, $\Gamma(F_{i+1}, B_{j-1}) = F_{i+1}(V) \wedge TR(V,V') \wedge B_{j-1}(V') \Rightarrow F_{i+1}(V) \wedge B_j(V)$. Since the right hand side is unsatisfiable, so is $\Gamma(F_{i+1}, B_{j-1})$.

The proof for a backward strengthening step is similar: in this case $B_{j+1} = B_{j+1} \wedge \mathrm{BI}(F_i, B_j)$. By the properties of an interpolant, we know that $\mathrm{BI}(F_i, B_j) \wedge F_i \equiv \bot$. Therefore, after the update $F_i \wedge B_{j+1} \equiv \bot$. Again, this implies that $\Gamma(F_{i-1}, B_{j+1}) \equiv \bot$: Recall that $F_{i-1}(V) \wedge TR(V,V') \Rightarrow F_i(V')$. Therefore $\Gamma(F_{i-1}, B_{j+1}) = F_{i-1}(V) \wedge TR(V,V') \wedge B_{j+1}(V') \Rightarrow F_i(V) \wedge B_{j+1}(V)$. Hence, unsatisfiability of the right hand side implies that which implies that $\Gamma(F_{i-1}, B_{j+1})$ is also unsatisfiable. □ □

Again, the indices are restricted since F_{n+1} and B_{n+1} are not yet defined. Moreover, the strengthening maintains the properties of a FRS and a BRS:

Lemma 4.3.8. *Let $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ be the result of a forward or backward strengthening step at (i,j) s.t. $i + j = n$. Then $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$ remain a FRS and a BRS resp.*

Proof. We show that $\bar{F}_{[n]}$ remains a FRS. The proof for $\bar{B}_{[n]}$ is similar. Recall that $\bar{F}_{[n]}$ was updated by setting $F_{i+1} = F_{i+1} \wedge \mathrm{FI}(F_i, B_j)$. The other components have not changed. It therefore suffices to show that after this update the following properties still hold: (1) $F_{i+1} \Rightarrow p$, (2) $F_i(V) \wedge TR(V,V') \Rightarrow F_{i+1}(V')$, and (3) $F_{i+1}(V) \wedge TR(V,V') \Rightarrow F_{i+2}(V')$. For the sake of the proof, we denote by F^o_{i+1} the old copy of F_{i+1}, before the update. We therefore have that $F_{i+1} = F^o_{i+1} \wedge \mathrm{FI}(F_i, B_j)$, and in particular $F_{i+1} \Rightarrow F^o_{i+1}$.

1. $F_{i+1} \Rightarrow p$: holds since $F_{i+1} \Rightarrow F^o_{i+1} \Rightarrow p$.

2. $F_i(V) \wedge TR(V,V') \Rightarrow F_{i+1}(V')$: holds since $F_i(V) \wedge TR(V,V') \Rightarrow F^o_{i+1}(V')$. In addition, by the properties of an interpolant, $F_i(V) \wedge TR(V,V') \Rightarrow \mathrm{FI}(F_i, B_j)(V')$. Conjoining the two implications, we conclude that $F_i(V) \wedge TR(V,V') \Rightarrow F^o_{i+1}(V') \wedge \mathrm{FI}(F_i, B_j)(V') = F_{i+1}(V')$.

3. $F_{i+1}(V) \wedge TR(V,V') \Rightarrow F_{i+2}(V')$: holds since $F_{i+1} \Rightarrow F^o_{i+1}$, which implies that $F_{i+1}(V) \wedge TR(V,V') \Rightarrow F^o_{i+1}(V) \wedge TR(V,V') \Rightarrow F_{i+2}(V')$.

□ □

Lemma 4.3.7 and Lemma 4.3.8 imply that if $\Gamma(F_i, B_j) \equiv \bot$ for *some* $0 \leq i,j \leq n$ s.t. $i+j=n$, then by iterating the forward and backward strengthening steps, we can eventually ensure that $\Gamma(F_i, B_j) \equiv \bot$ for *every* $0 \leq i,j \leq n$ s.t. $i+j=n$, and in particular for $i=0, j=n$ and $i=n, j=0$. Thus, we apply an *iterative local strengthening* starting from (i,j), strengthening and extending $\bar{F}$ and $\bar{B}$:

Definition 4.3.9 (Iterative Local Strengthening)**.** Let $0 \leq i,j \leq n$ be indices s.t. $i+j=n$ and $\Gamma(F_i, B_j) \equiv \bot$. *Iterative local strengthening from* (i,j) performs:

1. Forward strengthening steps starting at (i,j), proceeding forward while increasing i and decreasing j until $(n-1,1)$ (strengthening $F_{i+1}, \ldots, F_n$), and

2. Backward strengthening steps starting at (i,j), proceeding backward while increasing j and decreasing i until $(1,n-1)$ (strengthening $B_{j+1}, \ldots, B_n$), and

3. Finally, once $\Gamma(F_n, B_0) \equiv \bot$, F_{n+1} is initialized by $\mathrm{FI}(F_n, B_0)$. Similarly, once $\Gamma(F_0, B_n) \equiv \bot$, B_{n+1} is initialized by $\mathrm{BI}(F_0, B_n)$.

Iterative local strengthening from (i, j) strengthens $F_{i+1}, \ldots, F_n$ and initializes F_{n+1}. Similarly, it strengthens $B_{j+1}, \ldots, B_n$ and initializes B_{n+1}. Lemma 4.3.7 ensures that forward and backward propagation of the strengthening steps is possible, and Lemma 4.3.8 ensures that a FRS and a BRS are obtained by strengthening.

Lemma 4.3.10. *Let $0 \leq i, j \leq n$ be indices s.t. $i+j = n$ and $\Gamma(F_i, B_j) \equiv \bot$. Iterative local strengthening from (i, j) terminates with a FRS and a BRS of length $n + 1$.*

Proof. Termination of iterative local strengthening follows from Lemma 4.3.8 and Lemma 4.3.7 that ensure that after a local strengthening step, we still have a FRS and a BRS, and in addition, the unsatisfiability of the local reachability check is propagated one step forward or backward. This ensures that the process can continue until eventually $\Gamma(F_n, B_0) \equiv \bot$, and $\Gamma(F_0, B_n) \equiv \bot$, allowing to compute F_{n+1} and B_{n+1}. Lemma 4.2.3 ensures that the initialization of F_{n+1} and B_{n+1} using $\mathrm{FI}(F_n, B_0)$ and $\mathrm{BI}(F_0, B_n)$ maintains the properties of a FRS and a BRS. This is because $F_n \wedge TR \Rightarrow \mathrm{FI}(F_n, B_0)' = F'_{n+1}$ and in addition $F_{n+1} \wedge \neg p = \mathrm{FI}(F_n, B_0) \wedge B_0 \equiv \bot$. Similarly, $B_{n+1} = \mathrm{BI}(F_0, B_n) \Leftarrow TR \wedge B'_n$ and in addition $B_{n+1} \wedge INIT = \mathrm{BI}(F_0, B_n) \wedge F_0 \equiv \bot$. □ □

Iterative local strengthening uses the BRS for the strengthening of the FRS and vice versa, demonstrating how each of them is used to make the other over-approximation tighter. The complete local strengthening procedure is described in Figure 4.2.

4.3.2.2 Global Strengthening Phase

We now consider the case where $\Gamma(F_i, B_{n-i}) \not\equiv \bot$ for every $0 \leq i \leq n$ in $\bar{F}_{[n]}$ and $\bar{B}_{[n]}$. By Lemma 4.3.5, this means that there is a real transition between every pair of consecutive sets in the aCEX Π, making local strengthening inapplicable since the aCEX is not locally invalid. Clearly this does not

```
17: function LocStrengthen(F̄, B̄, n)
18:     i = FindStrengthen(F̄, B̄, n)
19:     if i == −1 then
20:         // No local strengthening
21:         // point was found
22:         //Move to GlbStrengthen
23:         return false
24:     else
25:         IterLS(F̄, B̄, n, i, n − i)
26:         return true
27:     end if
28: end function
```

(a) Local Strengthening

```
29: function IterLS(F̄, B̄, n, i, j)
30:     while i < n do
31:         F_{i+1} = F_{i+1} ∧ FI(F_i, B_{n−i})
32:         i = i + 1
33:     end while
34:     F̄.Add(FI(F_n, B_0))
35:     while j < n do
36:         B_{j+1} = B_{j+1} ∧ BI(F_{n−j}, B_j)
37:         j = j + 1
38:     end while
39:     B̄.Add(BI(F_0, B_n))
40: end function
```

(b) Iterative Local Strengthening

Figure 4.2: Local strengthening procedures

imply that the aCEX is valid, and further checks are needed. We therefore turn to examine the (in)validity of the aCEX in a more global manner.

Similarly to the principle used in CEGAR [19] for an *abstract* counterexample, here too, if the aCEX Π is invalid, there exists a minimal index $i \leq n+1$ representing the minimal prefix of the aCEX that has no matching path in M.

The idea behind checking the validity of an aCEX $\Pi_0 \rightarrow \ldots \rightarrow \Pi_{n+1}$ in

a global manner is therefore to search for such a minimal index where the aCEX becomes invalid, if such an index exists. In order to find a matching counterexample for $\Pi_0 \rightarrow \ldots \rightarrow \Pi_{n+1}$ in M or ensure that none exists, we therefore search for an index where the aCEX becomes invalid, if it exists. Since in our case the aCEX is known not to be locally invalid, we know that the prefix Π_0, Π_1 is necessarily valid. The search therefore starts from the prefix X_0, X_1, X_2 and goes forward gradually, while extending the prefix. In the i-th step (starting from $i = 2$), we consider the prefix $\Pi_0, \ldots, \Pi_i$. The goal is to check if $\Pi_0 \wedge TR \wedge \Pi_1' \wedge TR \wedge \Pi_2'' \wedge \ldots \wedge TR \wedge \Pi_i^{\langle i \rangle}$ (*) is satisfiable, meaning that a matching path to this prefix of the aCEX exists in M.

Recall that for $i \leq n$, (*) is actually the formula $INIT \wedge TR \wedge (F_1 \wedge B_n)' \wedge TR \wedge (F_2 \wedge B_{n-1})'' \wedge \ldots \wedge TR \wedge (F_i \wedge B_{n-i+1})^{\langle i \rangle}$. For $i = n + 1$ the last conjunct consists of B_0 only (without an $\bar{F}$-component). In fact, since in a FRS $F_j \wedge TR \Rightarrow F_{j+1}'$, then removing all $\bar{F}$ components except for the first (*INIT*) results in an equivalent formula. Similarly, since in a BRS $B_{j+1} \Leftarrow TR \wedge B_j'$, removing all $\bar{B}$ components but the last (B_{n-i+1}) again results in an equivalent formula. This simplifies the formula as follows.

Lemma 4.3.11. *For every* $2 \leq i \leq n+1$*:* $\Pi_0 \wedge \mathrm{TR}(V, V') \wedge \Pi_1' \wedge \mathrm{TR}(V', V'') \wedge \Pi_2'' \wedge \ldots \wedge \mathrm{TR}(V^{i-1}, V^i) \wedge \Pi_i^{\langle i \rangle}$ *is equivalent to* $\mathrm{INIT} \wedge \mathrm{TR}(V, V') \wedge \ldots \wedge \mathrm{TR}(V^{i-1}, V^i) \wedge B_{n-i+1}^{\langle i \rangle}$.

Proof. Similarly to the proof of Lemma 4.3.5, it can be shown that $INIT \wedge TR \wedge (F_1 \wedge B_n)' \wedge TR \wedge (F_2 \wedge B_{n-1})'' \wedge \ldots \wedge TR \wedge (F_i \wedge B_{n-i+1})^{\langle i \rangle}$ is equivalent to $INIT \wedge TR \wedge TR \wedge \ldots \wedge TR \wedge B_{n-i+1}^{\langle i \rangle}$, where the removal of the $\bar{F}$-components is due to the property $F_j \wedge TR \Rightarrow F_{j+1}'$ of a FRS, and the removal of the $\bar{B}$-components is due to the property $B_{j+1} \Leftarrow TR \wedge B_j$ of a BRS. A similar equivalence holds for $i = n + 1$, when the last conjunct in the formula is $B_0^{\langle n+1 \rangle}$ (without an $\bar{F}$-component). □ □

DAR therefore checks formulas of the form $INIT \wedge TR \wedge \ldots \wedge TR \wedge B_{n-i+1}^{\langle i \rangle}$ starting from $i = 2$. It keeps on adding transitions until either the formula

becomes unsatisfiable, or until $i = n + 1$ is reached (ending with $B_0 = \neg p$). If the formula is still satisfiable for $i = n + 1$, a counterexample of length $n + 1$ is found and DAR terminates.

If for some $2 \leq i \leq n+1$, $INIT \wedge TR \wedge \ldots \wedge TR \wedge B_{n-i+1}^{\langle i \rangle}$ turns out to be unsatisfiable, making the aCEX invalid, then first $\bar{F}_{[n]}$ is strengthened:

Definition 4.3.12. Let $INIT \wedge TR \wedge \ldots \wedge TR \wedge B_{n-i+1}^{\langle i \rangle} \equiv \bot$ for some $2 \leq i \leq n+1$, and let $\langle I_0, I_1, \ldots, I_{i+1} \rangle$ be an interpolation-sequence for $\langle A_1 = INIT \wedge TR, A_2 = TR, \ldots, A_i = TR, A_{i+1} = B_{n-i+1}^{\langle i \rangle} \rangle$. A *global strengthening step at index* i strengthens F_j for every $1 \leq j \leq \min\{i, n\}$ by setting $F_j = F_j \wedge I_j$.

The condition $1 \leq j \leq \min\{i, n\}$ ensures that if $i = n + 1$, strengthening is applied only up to F_n since F_{n+1} is not yet defined[1]. The following Lemma, along with Lemma 4.3.5 ensures that after a global strengthening step, the strengthened aCEX is locally invalid.

Lemma 4.3.13. *Let $\bar{F}_{[n]}$ be the result of a global strengthening step at index $2 \leq i \leq n + 1$. Then $\bar{F}_{[n]}$ remains a FRS. In addition, $\Gamma(F_{i-1}, B_{n-i+1}) \equiv \bot$.*

Proof. We first show that $\bar{F}_{[n]}$ remains a FRS. Since the F_j's are only strengthened, the implications $F_j \Rightarrow p$ still hold for every $0 \leq j \leq n$. We now show that $F_j \wedge TR \Rightarrow F'_{j+1}$ still holds as well for every $0 \leq j \leq n - 1$. For $i + 1 \leq j \leq n - 1$ no proof is needed since F_j and F_{j+1} have not changed in this case. We now consider $0 \leq j \leq \min\{i, n - 1\}$. Prior to strengthening, it was the case that (1) $F_j \wedge TR \Rightarrow F'_{j+1}$ (a property of a FRS). In addition, for $j = 0$, $I_0 \wedge A_1 \Rightarrow I_1$, meaning that $\top \wedge INIT \wedge TR \Rightarrow I'_1$ (a property of an interpolation-sequence). Therefore $F_0 \wedge TR = \top \wedge INIT \wedge TR \Rightarrow I'_1$. Conjoining this with (1), implies that $F_0 \wedge TR \Rightarrow (F_1 \wedge I_1)'$. Similarly, for $0 < j \leq \min\{i, n - 1\}$, $I_j \wedge A_{j+1} \Rightarrow I'_{j+1}$, meaning that $I_j \wedge TR \Rightarrow I'_{j+1}$ (a property of an interpolation-sequence). Therefore, by conjoining this implication with (1) we get that for $0 < j \leq \min\{i, n - 1\}$, $(F_j \wedge I_j) \wedge TR \Rightarrow$

[1] If a global strengthening step is performed at $i = n + 1$, then F_{n+1} can be initialized to I_{n+1}.

$(F_{j+1} \wedge I_{j+1})' \Rightarrow F'_{j+1}$. Therefore, the implications required by a FRS still hold after the updates of F_j to $F_j \wedge I_j$ for $0 < j \leq \min\{i, n\}$.

We now show that $\Gamma(F_{i-1}, B_{n-i+1}) \equiv \bot$. By the definition of an interpolation-sequence, $I_i \wedge A_{i+1} \Rightarrow I_{i+1}$ where $I_{i+1} \equiv \bot$, which means that $\bot \equiv I_i \wedge A_{i+1} = I_i \wedge B_{n-i+1}$. In addition, $I_{i-1} \wedge A_i \Rightarrow I_i$, meaning that $I_{i-1} \wedge TR \Rightarrow I'_i$. Thus, due to the strengthening of F_{i-1} by conjoining it with I_{i-1}, we conclude that after the update $F_{i-1} \Rightarrow I_{i-1}$ and hence $F_{i-1} \wedge TR \Rightarrow I_{i-1} \wedge TR \Rightarrow I'_i$. Along with the property that $I_i \wedge B_{n-i+1} \equiv \bot$, we conclude that $\Gamma(F_{i-1}, B_{n-i+1}) = F_{i-1} \wedge TR \wedge B'_{n-i+1} \Rightarrow I'_i \wedge B'_{n-i+1} \equiv \bot$. □ □

DAR now uses iterative local strengthening from $(i-1, n-i+1)$ (Definition 4.3.9) to strengthen $F_i, \ldots, F_n$ and $B_{n-i+2}, \ldots, B_n$[2], as well as initialize F_{n+1} and B_{n+1}. The complete global strengthening procedure is described in Figure 4.3.

4.3.3 Correctness of DAR

Having described all the components of DAR, we now return to the proof of Theorem. 4.3.1, which ensures that DAR terminates with a correct answer.

Theorem. 4.3.1. We first consider iteration n. The iteration performs a finite number of operations (as described in the local and global strengthening phases), and hence it necessarily terminates. Moreover, it terminates successfully while extending the FRS and the BRS iff no counterexample of length n exists. This can be seen since a counterexample is reported in the global strengthening phase when the formula $INIT \wedge TR \wedge TR \wedge \ldots \wedge TR \wedge B_0^{\langle n \rangle} =$

[2]Note that instead of performing a local strengthening of $\bar{B}$ as part of the iterative local strengthening, an interpolation-sequence $\langle J_0, J_1, \ldots, J_{i+1} \rangle$ for $\langle A_1 = TR \wedge B_{n-i}^{\langle i+1 \rangle}, A_2 = TR, \ldots A_i = TR, A_{i+1} = INIT \rangle$ can be used to strengthen $B_{n-i+1}, \ldots, B_n$ by setting $B_{n-i+j} = B_{n-i+j} \wedge J_j$ for $1 \leq j \leq i$, and to initialize B_{n+1} to J_{i+1}. In this case, iterative local strengthening will be performed only forward, updating $\bar{F}$ only. For simplicity of the presentation, we use iterative local strengthening both forward and backward instead of using an interpolation-sequence for the backward update.

```
41: function GlbStrengthen(F̄, B̄, n)
42:     for i = 2 → n + 1 do        // n = 0 does not go into the loop
43:         if INIT ∧ TR ... ∧ TR ∧ B^⟨i⟩_{n−i+1} == UNSAT then
44:             Ī = GetInterpolationSeq()
45:             for j = 1 → min{i, n} do
46:                 F_j = F_j ∧ I_j
47:             end for
48:             IterLS(F̄, B̄, n, i − 1, n − i + 1)
49:             return true
50:         end if
51:     end for
52:     return false        // counterexample
53: end function
```

Figure 4.3: Global strengthening procedure

$INIT \wedge TR \wedge \ldots \wedge TR \wedge \neg p^{\langle n \rangle}$ is satisfiable (see line 43 in Figure 4.3), thus a satisfying assignment to it provides a real counterexample. (An additional case happens in the first iteration, when $n = 0$ and the formula $INIT \wedge TR \wedge \neg p$ is satisfiable). In all other cases, the FRS and BRS are extended successfully by iterative local strengthening (see Lemma 4.3.10) and by Lemma 4.2.5, this ensures that no counterexample of length n exists.

Thus, if a counterexample of length n exists, then DAR will find it and will terminate at iteration n at latest reporting a counterexample (lines 3 and 10 in Figure 4.1). It might terminate earlier if a shorter counterexample is found, but it cannot terminate with a "Verified" result, since this only happens if fixpoint is reached, in which case by Lemma 4.2.7, $M \models AGp$ in contradiction to the existence of a counterexample.

On the other hand, if $M \models AGp$, all iterations of DAR terminate successfully computing a FRS and a BRS (since there is no counterexample of any length). For every n, at the end of iteration n, if no fixpoint is reached then in particular F_{n+1} includes at least one state that is not in $\bigvee_{i=0}^{n} F_i$. Now consider $N = 2^{|V|} + 1$. N is well-defined since V is finite (as M is). Since 2^V

is the set of all states in M and $N = 2^{|V|} + 1$, at iteration N at latest all the states of M are already in $\bigvee_{i=0}^{N} F_i$, and a fixpoint must be reached. In this case, DAR terminates and returns "Verified" (see line 7 in Figure 4.1). □

□

4.4 Experimental Results

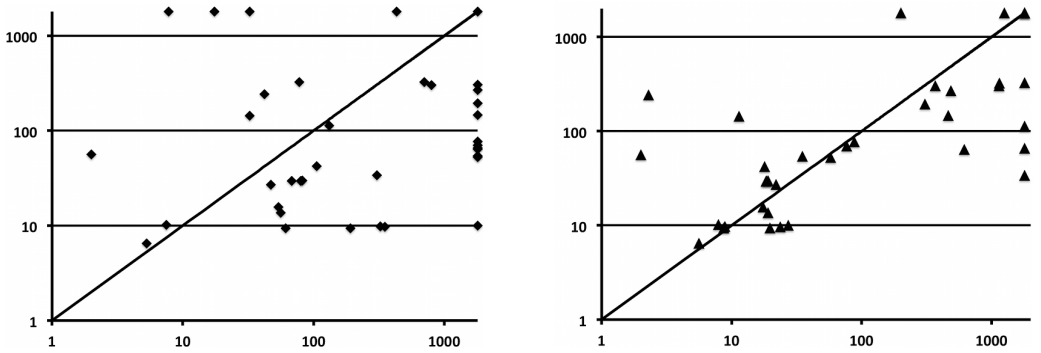
To implement DAR we collaborated with *Jasper Design Automation*[3]. We measured the efficiency of DAR by comparing it against two top-tier methods: ITP and IC3. We used Jasper's formal verification platform in order to implement DAR, ITP and IC3. Collaborating with Jasper allowed us to experiment with various real-life industrial designs and properties from various major semiconductor companies.

Our implementations use known optimizations for the checked methods (e.g. [14, 28]) and are comparable to other optimized implementations available online. For DAR we used some basic procedures to simplify the computed interpolants when possible. Our implementation of DAR is preliminary and can be further optimized.

For the experiments we used 37 real safety properties from real industrial hardware designs. The timeout was set to 1800 seconds and experiments were conducted on systems with Intel Xeon X5660 running at 2.8GHz and 24GB of main memory.

Table 4.1 shows different parameters for all three algorithms on various industrial examples. The parameters presented are: *time* is the runtime in seconds; *depth* represents the number of over-approximated sets of states computed when the algorithm converges (for ITP, the number of sets computed for the last bound used, and for DAR, the length of $\bar{F}$ and $\bar{B}$); *maximum unrolling* is shown for ITP and DAR (IC3 does not use unrolling) and represents the maximum unrolling used during verification; and for DAR

[3]An EDA company: http://www.jasper-da.com



(a) Runtime DAR vs. IC3. (b) Runtime DAR vs. ITP.

Figure 4.4: Y-axis represents DAR's runtime in seconds. X-axis represents runtime in seconds for the compared algorithm (IC3 or ITP). Points below the diagonal are in favor of DAR.

we present ♯GS and GS_R, which are the number of iterations where *Global Strengthening* was used and the ratio between the global strengthening to the total number of iterations. ♯GS also indicates the number of iterations where *local strengthening* was insufficient (requiring to use global strengthening and therefore unrolling).

Examining the results shows that the use of unrolling in DAR is indeed limited and that *local strengthening* plays a major part during verification, with $\mathrm{GS}_R < 0.5$ in most cases, indicating that local strengthening is often sufficient. Moreover, even when unrolling is used, its depth is usually smaller compared to the convergence depth, as indicated by maximum unrolling. Note that the maximum unrolling provides an *upper bound* on the unrolling, and the actual unrolling can be smaller in some global strengthening phases. For falsified properties (counterexample exists) unrolling is necessarily applied up to the length of the counterexample in the last iteration. Yet, in many cases local strengthening is still sufficient in previous iterations.

Another conclusion from the table is that a lower depth of convergence does not necessarily translate to a better runtime. We can see that in many cases, while ITP converges with less computed sets it takes more time than

DAR. This is not surprising since the number of computed sets presented for ITP considers only the sets computed in the last bound that was used, disregarding sets from previous bounds. The same can be seen with regards to IC3. While IC3 converges at a lower depth (on some cases), it still does not necessarily perform better. This is mainly due to the different effort invested by each algorithm in the strengthening and addition of a new over-approximated set.

Figure 5.11 shows a runtime comparison between DAR and IC3 (Figure 4.4a) and ITP (Figure 4.4b) on all 37 industrial examples, including those from Table 4.1. In 19 out of 37 cases, DAR outperforms ITP, and in 25 out of 37 cases it outperforms IC3. In 18 out of 37 cases DAR outperforms both methods. DAR could not solve only 5 cases, whereas ITP and IC3 failed to solve 7 and 12 cases respectively. The overall performance, when summarized, is in favor of DAR with 36% improvement in run time when compared to ITP and 52% improvement when compared to IC3.

Cases where DAR outperforms ITP can be explained by the following factors. First, DAR avoids unrolling when not needed, therefore its SAT calls are simpler. Second, DAR uses over-approximated sets computed in early iterations and strengthens them as needed, while ITP does not re-use sets that were computed for lower bounds and restarts its computation when a spurious counterexample is encountered. Cases where DAR outperforms IC3 are typically when DAR's strengthening is more efficient than IC3's inductive generalization, requiring less computation power at each iteration.

Since DAR relies heavily on interpolants, the cases where DAR performs worse than IC3 are usually those where the interpolants grow large and contain redundancies. This is also true when comparing to ITP. Since DAR computes more interpolants than ITP and also accumulates them, it is more sensitive to the size of the computed interpolants.

We also used the HWMCC'11 benchmark in our experiments. While there are a lot of cases where all methods perform the same, there are also

examples where DAR outperforms both IC3 and ITP (some are shown at the bottom of Table 4.1). The benchmark also includes examples where IC3 or ITP perform better than DAR. The majority of these cases are simple and solved in a few seconds.

			IC3		ITP			DAR				
Name	♯Vars	Status	D	Time[s]	D	MaxU	Time[s]	D	MaxU	♯GS	GS_R	Time[s]
Ind_1	11854	true	46	799	41	28	1138	49	35	21	0.42	**303**
Ind_2	11854	true	44	701	41	28	1148	49	35	18	*0.36*	**326**
Ind_3	11866	true	11	82	5	2	**19.1**	11	8	4	*0.33*	29.9
Ind_4	11877	true	NA	TO	33	12	307	36	30	18	0.48	**194**
Ind_5	11871	false	NA	TO	NA	20	88	19	20	10	0.5	**77**
Ind_6	11843	false	NA	TO	NA	19	77	18	19	9	0.47	**70**
Ind_7	1247	true	6	**1.5**	3	2	2	17	5	9	0.5	56.3
Ind_8	1247	true	7	**7.8**	17	23	1250	NA	NA	NA	NA	TO
Ind_9	449	true	337	**78**	NA	NA	TO	45	12	22	0.48	327
Ind_{10}	331	true	458	305	NA	NA	TO	26	11	15	0.56	**33.9**
Ind_{11}	330	true	419	132	NA	NA	TO	38	12	19	0.49	**113**
Ind_{12}	450	true	22	**32.5**	NA	NA	TO	NA	NA	NA	NA	TO
Ind_{13}	3837	false	NA	TO	NA	68	369	67	68	33	0.48	**305**
Ind_{14}	3837	false	NA	TO	NA	69	487	68	69	25	*0.36*	**269**
Ind_{15}	3836	true	6	42	4	2	**2.3**	70	64	32	0.45	243
Ind_{16}	11860	true	9	32.5	5	2	**11.4**	33	32	16	0.47	144
Ind_{17}	11878	true	14	68	7	4	**18.4**	11	8	4	*0.33*	29.5
Ind_{18}	3836	true	NA	TO	6	17	27.3	15	6	6	*0.37*	**10**
intel007	1307	true	5	**53.5**	NA	NA	TO	NA	NA	NA	NA	TO
intel018	491	true	NA	TO	57	35	695	78	51	33	0.42	**64**
intel019	510	true	NA	TO	52	35	515	96	57	43	0.44	**310**
intel023	358	true	NA	TO	NA	NA	TO	86	53	35	*0.4*	**66**
intel026	492	true	53	47.1	50	35	**21.9**	70	51	34	0.48	27.8

Table 4.1: Parameters of the experiments. *Name*: name of the verified property; ♯*Vars*: number of state variables in the cone of influence; *Status*: *true* - verified property, *false* - indicates a counterexample; D: *convergence depth* representing the number of over-approximated sets of states computed when the algorithm converges (for ITP, the number of sets computed for the last bound used, and for DAR, the length of $\bar{F}$ and $\bar{B}$); *MaxU*: *maximum unrolling* used during verification; ♯*GS*: number of times Global Strengthening is used in DAR; GS_R: ratio between iterations using global strengthening to the total number of iterations; *Time[s]*: *time* in seconds. Minimal runtime appears in boldface. Properties above the full line are from real industrial designs. Those underneath the line are from HWMCC'11.

Chapter 5

Efficient Generation of Small Interpolants in Conjuctive Normal Form

The work presented in this chapter appeared in [60]. This chapter describes a novel approach for interpolant computation in the context of SAT-based model checking. The main contribution of this work is the ability to produce *small* interpolants in *Conjunctive Normal Form* (CNF) efficiently. In order to compute an interpolant, our work takes advantage both of the properties of the resolution refutation, generated by the SAT solver, and of the structure of the model checking problem at hand. Another contribution of this work is the algorithm *CNF-ITP*, which is an enhanced version of the original interpolation-based model checking algorithm [43] (ITP). CNF-ITP exploits of the fact that interpolants are given in CNF.

Interpolants are used in various domains. Here, like in previous chapters, we focus on ITP. In his seminal work [43], McMillan presents a recursive procedure for interpolant generation from a proof. The procedure initially assigns a propositional formula to each one of the leaves in the resolution refutation (hypothesis clauses). It then recursively assigns a propositional

formula to every node in the refutation by either conjoining or disjoining the propositional formulas of its predecessors. Choosing between conjunction or disjunction depends on whether the pivot variable is local to $A(X, Y)$ or not. The formula that is assigned for the empty clause represents the interpolant.

While this algorithm is linear in the size of the proof, the resulting interpolant is a non-CNF propositional formula that mirrors the structure of the resolution refutation. Thus, when the resolution refutation is large, so is the interpolant. Moreover, the resulting formula is often highly redundant, meaning that the interpolant can be simplified and be represented by an equivalent smaller formula.

ITP (Section 2.4) requires the interpolants to be fed back into the SAT solver for computing the next interpolant. Therefore, in those cases where the size of interpolants is large, the resulting SAT problem may be intractable.

We strive to solve this problem by directly generating small interpolants in CNF. One way to compute an interpolant is by *existential quantification*. Considering the unsatisfiable formula $A(X, Y) \wedge B(Y, Z)$, $I(Y) = \exists X(A(X, Y))$ is an interpolant. For a CNF formula $A(X, Y)$, $\exists X(A(X, Y))$ can be created by iteratively applying variable elimination[1] on X variables in $A(X, Y)$. The problem with this approach is that variable elimination is exponential, and, therefore impractical, given a large set of variables.

In this work, we provide a novel resolution-refutation-guided method for variable elimination in order to derive an interpolant in CNF. This procedure, while creating less clauses than naïve variable elimination procedures, might still result in an exponential blow-up.

Our solution is first to build an *approximated* interpolant $I_w(Y)$ for which $I_w(Y) \wedge B(Y, Z)$ may be satisfiable. We refer to such an interpolant as a B_{weak}*-interpolant*. Computing the B_{weak}-interpolant is based on the method of resolution-refutation-guided variable elimination but is much more effi-

[1] *Variable elimination* [24] is an operation that eliminates all occurrences of a variable v from a CNF formula by replacing clauses containing v with the result of pairwise resolutions between all clauses containing the literal v and those containing the literal $\neg v$

cient. The second stage of our method aims at strengthening $I_w(Y)$ and transforming it into an interpolant $I(Y)$ where $I(Y) \wedge B(Y,Z)$ is unsatisfiable. We refer to this process as *B-Strengthing*.

In order to transform a B_{weak}-interpolant into an interpolant we need to make sure that $A(X,Y) \Rightarrow I_w(Y)$ and that $I_w(Y) \wedge B(Y,Z)$ is unsatisfiable. This can be done by finding all satisfying assignments $s(Y)$ to $I_w(Y) \wedge B(Y,Z)$ and conjoining $\neg s(Y)$ with $I_w(Y)$. Clearly, since $A(X,Y) \wedge B(Y,Z)$ is unsatisfiable, $A(X,Y) \Rightarrow \neg s(Y)$ for all such assignments. Note that an assignment s is a conjunction of literals, and therefore its negation is a clause. By this we keep $I_w(Y)$ in CNF. The number of such assignments may be vast, and therefore this is an inefficient method.

To overcome this, instead of adding a clause to $I_w(Y)$ we *generalize* it to a sub-clause so as to block a larger set of assignments. In order to perform an efficient generalization we use the structure of A. In the context of model checking, $A(V,V') = Q(V) \wedge TR(V,V')$ where V is the set of variables in the checked system and TR is the transition relation. Using this fact allows us to perform *inductive generalization* [8].

We implemented CNF-ITP, a model checking algorithm which is a variant of ITP [43], but which uses the above method to compute the interpolants. Our goal was to measure the impact of our interpolant computation method on the underlying model checking algorithm. However, CNF-ITP also exploits the fact that interpolants are given in CNF in order to improve the traditional ITP. Our improvements to ITP were inspired by [8].

For the experiments we used the HWMCC'12 benchmark set. The interpolants computed by our method, compared to those computed by the original ITP algorithm of [43], were much smaller in size in the vast majority of cases. Sometimes, the size was up to *two* orders of magnitude smaller. Our procedure significantly outperformed ITP and solved some test cases that ITP could not solve. To complete our experiments, we also compared CNF-ITP to the successful IC3 [8] algorithm. We found that CNF-ITP out-

performed IC3 [8] in a large number of cases.

5.0.1 Related Work

A well-known problem of interpolants is their size. Several works try to deal with this problem. The work in [14] suggests dealing with the increasing size of interpolants by using circuit compaction. While this process can be efficient in some cases, it may consume considerable resources for very large interpolants. Moreover, compacting an interpolant does not result in a CNF formula, whereas our approach results in interpolants in CNF.

As we have already noted, an interpolant computed from a resolution refutation mirrors its structure. Several works [2, 52] deal with reductions to the resolution refutation. Since our method uses resolution refutation it too can benefit from such an approach.

During interpolant computation, our approach only uses the relevant parts of the resolution refutation. The idea of holding and maintaining only the relevant parts of the resolution derivation was proposed and proved useful in [53] in the context of group-oriented minimal unsatisfiable core extraction.

Deriving interpolants in CNF was suggested in [38]. The authors suggest applying a set of reordering rules for resolution refutations so that the resulting interpolant will be in CNF. As the authors state in the paper, the described procedure does not always return an interpolant in CNF. Also, the reordering of a resolution refutation may result in an exponential blow up of the proof and, as stated in [25], reordering is not always possible. In contrast to [38], our method does not rewrite the resolution refutation generated by the SAT solver.

The work in [17] suggests an interpolant computation method that does not use the generated resolution refutation. In addition, an interpolant that results from the use of that method is in a Disjunctive Normal Form (DNF). Our work, on the other hand, uses the resolution refutation and generates interpolants in CNF efficiently.

5.1 Preliminaries

For a model M, with a slight abuse of notation, we sometimes refer to a propositional formula over V as a set of states in M. We treat CNF formulas as sets of clauses and use a union of sets to denote a conjunction of two CNF formulas. In addition, given a sequence of clauses π, we will use $\alpha \in \pi$ to denote a clause α that is part of the sequence π.

Recall that for a formula X, $\text{Vars}(X)$ is the set of variables appearing in X.

Definition 5.1.1 (Local and Global Variable)**.** Let (A, B) be a pair of formulas in CNF. A variable v is A-local (B-local) iff $v \in \text{Vars}(A) \backslash \text{Vars}(B)$ ($v \in \text{Vars}(B) \backslash \text{Vars}(A)$); v is (A, B)-global or, simply, global, iff $v \in \text{Vars}(A) \cap \text{Vars}(B)$.

We will use the notions of weaker versions of interpolants that fulfill two out of three interpolant properties (recall Definition 2.3.1).

Definition 5.1.2 (B_{weak}-Interpolant)**.** Let (A, B) be a pair of formulas in CNF such that $A \wedge B \equiv \bot$. The *B_{weak}-interpolant* for (A, B) is a formula I such that:

- $A \Rightarrow I$.
- $\text{Vars}(I) \subseteq \text{Vars}(A) \cap \text{Vars}(B)$.

Definition 5.1.3 (Non-Global-Interpolant)**.** Let (A, B) be a pair of formulas in CNF such that $A \wedge B \equiv \bot$. The *non-global-interpolant* for (A, B) is a formula I such that:

- $A \Rightarrow I$.
- $I \wedge B \equiv \bot$.

Recall that a resolution derivation π can naturally be conceived of as a directed acyclic graph (DAG) whose vertices correspond to all the clauses of π and in which there is an edge from a clause α_j to a clause α_i iff $\alpha_i = \alpha_j \otimes \alpha_k$. A clause $\beta \in \pi$ is a *parent* of $\alpha \in \pi$ iff there is an edge from β to α. A clause $\beta \in \pi$ is *backward reachable* from $\gamma \in \pi$ if there is a path (of 0 or more edges) from β to γ. The set of all vertices backward reachable from $\beta \in \pi$ is denoted $\Gamma(\pi, \beta)$.

For this work, we will need a definition of an A-resolution refutation, that is, a projection of a given resolution refutation π to the clause set A:

Definition 5.1.4 (A-Resolution Refutation)**.** Let $\pi = (\alpha_1, \alpha_2, \ldots, \square)$ be a resolution refutation of length n of the CNF formula $G = A \wedge B$. The *A-resolution refutation* π_A derived from π is defined as follows: $\pi_A = \pi_A^n$ where π_A^i is defined incrementally:

$$\pi_A^i = \begin{cases} \emptyset & i = 0 \\ \pi_A^{i-1} \cdot \alpha_i & \begin{cases} \alpha_i \in A \text{ or} \\ \alpha_i = \alpha_j \otimes^v \alpha_k \text{ such that } \alpha_j \in \pi_A^{i-1} \text{ or } \alpha_k \in \pi_A^{i-1} \end{cases} \end{cases}$$

In DAG terminology π_A is a sub-graph of π that contains only those vertices whose clauses belong to A, and the edges between such clauses. Note that a clause $\alpha \in \pi$ may have 0 or 2 parents, while a clause $\alpha \in \pi_A$ may also have 1 parent (if the second parent is implied only by the clauses of B).

Given a clauses set F, we denote the set of clauses containing the literal v and $\neg v$ as F_v^+ and F_v^- respectively. Given a CNF formula F and a variable $v \in \text{Vars}(F)$, *variable elimination* [24] is an operation that removes v from F by replacing clauses containing the variable v with the result of all pairwise resolution between F_v^+ and F_v^-. The resulting formula $VE(F, v)$ is equisatisfiable with F [24]. The DP algorithm for deciding propositional satisfiability [24] uses variable elimination until either the empty clause $\square$ is

derived, in which case the formula is unsatisfiable, or all the variables appear in one polarity only, in which case the formula is satisfiable. It is well known that the original DP algorithm suffers from exponential blow-up.

A bounded version of variable elimination has been an essential contributor to the efficiency of modern SAT preprocessing algorithms (that is, algorithms that truncate the size of the CNF formula before embarking on the search) since the introduction of the SatELite preprocesor [27]. In *bounded variable elimination*, used in SatELite, a variable v is eliminated iff the operation does not increase the number of clauses.

5.2 Generating Interpolant Approximation in CNF

In this section we propose a method for generating a B_{weak}-interpolant (recall Definition 5.1.2) in CNF. First, we describe two algorithms for generating interpolants in CNF. In practice, both algorithms are not applicable to all cases, because of exponential blow-up. Thereafter we introduce an efficient algorithm which is guaranteed to return a B_{weak}-interpolant in CNF, and which may for some cases return an interpolant in CNF.

We start with the following lemma which follows directly from resolution derivation properties. This lemma serves as the basis for our first algorithm.

Lemma 5.2.1. *Let π_A be an A-resolution refutation. Let $P(\alpha_i)$ for $\alpha_i \in \pi_A \setminus A$) be the set of (1 or 2) parents of α_i. Then, $P(\alpha_i) \wedge B \Rightarrow \alpha_i$*

Our first algorithm for generating an interpolant in CNF is based on naïve variable elimination. First it generates a resolution refutation of the given formula using a SAT solver. Then it initializes the interpolant by those clauses of A that are backward reachable from $\square$ (the empty clause). Note that at this stage I is a non-global-interpolant since it contains A-local variables (recall Definition 5.1.3). Finally, the algorithm gradually turns the non-global-

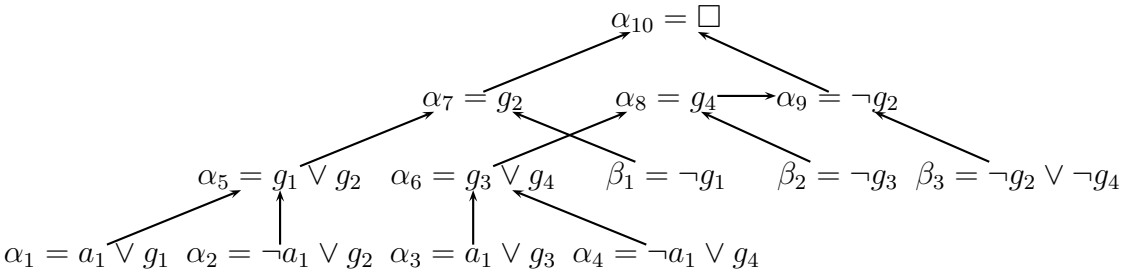


Figure 5.1: An example of a resolution refutation. Assume $A = \{\alpha_1, \ldots, \alpha_4\}$ and $B = \{\beta_1, \ldots, \beta_3\}$.

```
1: function ILVE(A, B)
2:     Generate a resolution refutation π using a SAT solver
3:     I_0 := A ∩ Γ(π, □)
4:     i := 0
5:     for all v ∈ Vars(I) ∩ Vars(A) do
6:         I_{i+1} := VE(I_i, v)
7:         i := i + 1
8:     end for
9:     return I_i
10: end function
```

Figure 5.2: Interpolant by A-Local Variables Elimination

interpolant into an interpolant by applying variable elimination over all A-local variables. Consider the example in Figure 5.1. Our algorithm would generate the following interpolant: $I = \{(g_1 \vee g_2), (g_1 \vee g_4), (g_3 \vee g_2), (g_3 \vee g_4)\}$. Unfortunately, the algorithm suffers from the same drawback as the DP algorithm [24]: exponential blow-up when variables keep being eliminated. The complete algorithm is presented in Figure 5.2

We show that Algorithm ILVE is correct.

Lemma 5.2.2. *Algorithm ILVE returns an interpolant.*

Proof. We will show that all the three interpolant properties hold for I:

1. $A \Rightarrow I$ holds, since all the clauses of I are generated by resolution over A clauses.

2. $I \wedge B \equiv \bot$. We prove the statement by induction on the number of iterations in the loop. For the first iteration it holds that $I_0 = (A \cap \Gamma(\pi, \Box)) \wedge B \equiv \bot$, since the structure of π implies that $\Box$ is reachable from $(A \cap \Gamma(\pi, \Box)) \wedge B$ ($A \cap \Gamma(\pi, \Box)$ are the leaves in π that belong to A, and $A \wedge B \equiv \bot$). Consider iteration $i+1$. By induction hypothesis it holds that $I_i \wedge B \equiv \bot$. By correctness of variable elimination [24], we have $VE(I_i \wedge B, v) \equiv \bot$. Note that $v \notin \text{Vars}(B)$, since v is A-local by construction, hence $VE(I_i \wedge B, v) \equiv VE(I_i, v) \wedge B \equiv \bot$, which proves our property, since I_{i+1} is exactly $VE(I_i, v)$.

3. $\text{Vars}(I) \subseteq \text{Vars}(A) \cap \text{Vars}(B)$ holds, since all the A-local variables are eliminated from I_0.

□

Recall that Algorithm 5.2 suffers from the same drawback as the DP algorithm [24]: exponential blow-up when variables are eliminated. We therefore move on to introduce our second attempt.

Our next algorithm is based on the observation that to eliminate a variable v from F, it is not necessary to apply resolution over all the pairs in F_v^+ and F_v^-, but rather only over those subsets that contribute to deriving a common ancestor in the resolution derivation. We need to introduce the notion of clause-interpolant.

Definition 5.2.3 (Clause-Interpolant)**.** Let (A, B) be an unsatisfiable pair of CNF formulas. Let α be a clause. Then, $I(\alpha)$ is a *Clause-Interpolant* of α iff:

- $A \Rightarrow I(\alpha)$
- $I(\alpha) \wedge B \Rightarrow \alpha$
- $\text{Vars}(I(\alpha)) \subseteq (\text{Vars}(A) \cap \text{Vars}(B)) \cup (\text{Vars}(A) \cap \text{Vars}(\alpha))$

A clause-interpolant is a generalization of an interpolant that allows one to associate an interpolant with every clause α in A-resolution refutation (recall Definition 5.1.4). As in the case of the standard interpolant, the clause-interpolant is implied by A. The conjunction of the clause-interpolant with B implies the clause α (instead of $\square$ for the standard interpolant). Finally, the clause-interpolant is allowed to contain in addition to global variables also A-local variables that appear in α. Note that a clause-interpolant of $\square$ is an interpolant.

Lemma 5.2.4. *A clause-interpolant of the empty clause $\square$ is an interpolant.*

Proof. Each of the three conditions in the definition of a clause-interpolant yields the corresponding condition in the definition of the interpolant. $\square$

The second algorithm for deriving an interpolant in CNF works as follows: it traverses the A-resolution refutation from the input clauses towards $\square$. It constructs a clause-interpolant for each traversed clause as follows. The clause-interpolant of each initial clause α is set to $\{\alpha\}$. For creating the clause-interpolant of a derived clause α, the algorithm first conjoins the clause-interpolants of α's parents. Then, if α was created by resolution over a local variable v, v is eliminated from the result by applying variable elimination (Figure 5.3, line 12). The clause-interpolant of $\square$ is returned as the interpolant. Consider again the example in Figure 5.1. We have $I(\alpha_5) = \alpha_1 \otimes^{a_1} \alpha_2 = g_1 \vee g_2$; $I(\alpha_6) = \alpha_3 \otimes^{a_1} \alpha_2 = g_3 \vee g_4$; $I(\alpha_7) = I(\alpha_5)$; $I(\alpha_9) = I(\alpha_8) = I(\alpha_6)$. Finally, the interpolant is $I(\square) = I(\alpha_7) \cup I(\alpha_9) = \{g_1 \vee g_2, g_3 \vee g_4\}$. Note that for our example, the interpolant generated by the current algorithm is smaller than the one generated by our previous algorithm, which applies exhaustive variable elimination. In practice, however, the current algorithm is not always scalable either, due to the same problem – exponential blow-up caused by variable elimination. Also note that for our simple example the intepolant comprises a cut $\{\alpha_5, \alpha_6\}$ in the A-resolution refutation, where all the clauses are implied by A only. One can show that

```
1: function IGCI(π_A = (α_1, α_2, ..., α_q, α_{q+1}, α_{q+2}, ..., α_p ≡ □))
2:     for all i ∈ {1, 2, ..., q} do
3:         I(α_i) := {α_i}
4:     end for
5:     for all i ∈ {q + 1, q + 2, ..., p ≡ □} do
6:         if α_i has exactly one parent β then
7:             I(α_i) := I(β)
8:         else
9:             if α_i = α_j ⊗^v α_k, where v is global then
10:                I(α_i) := I(α_j) ∪ I(α_k)
11:            else      // α_i = α_j ⊗^v α_k, where v is A-local
12:                I(α_i) := VE(I(α_j) ∪ I(α_k), v)
13:            end if
14:        end if
15:    end for
16:    return I(□)
17: end function
```

Figure 5.3: Interpolant Generation with Clause-Interpolants

whenever such a cut exists it comprises an interpolant. Unfortunately, in the general case such cuts do not usually exist. The complete algorithm appears in Figure 5.3.

We prove the algorithm's correctness starting with a straightforward lemma.

Lemma 5.2.5. *Let P_1 and P_2 be formulas in propositional logic. Then, $P_1 \Rightarrow P_2$ if and only if $P_1 \wedge \neg P_2 \equiv \bot$.*

We need to prove additional two lemmas.

Lemma 5.2.6. *Let I and B be formulas in CNF, α be a clause, and v be a variable, such that:*

1. $I \wedge B \Rightarrow \alpha$
2. $v \notin \mathit{Vars}(B \cup \alpha)$

Then, $VE(I, v) \wedge B \Rightarrow \alpha$.

Proof. By applying Lemma 5.2.5 over the lemma's first condition $I \wedge B \Rightarrow \alpha$ we have $I \wedge B \wedge \neg\alpha \equiv \bot$. By variable elimination properties [24] we have $VE(I \wedge B \wedge \neg\alpha, v) \equiv \bot$. The latter statement and the lemma's second condition $v \notin \text{Vars}(B \cup \alpha)$ imply $VE(I \wedge B \wedge \neg\alpha, v) \equiv VE(I, v) \wedge B \wedge \neg\alpha \equiv \bot$. Finally, by again applying Lemma 5.2.5 we have $VE(I, v) \wedge B \Rightarrow \alpha$. □

Lemma 5.2.7. *Let π be a resolution refutation and let π_A be the A-resolution input sequence for IGCI. Then, for every $I(\alpha_i)$ computed by IGCI, $I(\alpha_i)$ is a clause interpolant.*

Proof. The proof is by induction on the number of iterations of both loops.

For every clause $\alpha \in A$, $I(\alpha) = \{\alpha\}$. Clearly, $I(\alpha)$ meets the three requirements of clause-interpolant (Definition 5.2.3)

We omit the proofs for cases where a derived clause α_i has only one parent β, and where $\alpha_i = \alpha_j \otimes^v \alpha_k$, where v is global, since they are straightforward.

Consider the case where $\alpha_i = \alpha_j \otimes^v \alpha_k$, where v is A-local. By induction hypothesis we have $A \Rightarrow I(\alpha_j)$ and $A \Rightarrow I(\alpha_k)$, hence $A \Rightarrow I(\alpha_j) \cup I(\alpha_k)$. All the clauses created by variable elimination $VE(I(\alpha_j) \cup I(\alpha_k), v)$ are derived from $I(\alpha_j) \cup I(\alpha_k)$ using resolution, hence we have $A \Rightarrow VE(I(\alpha_j) \cup I(\alpha_k), v)$ and the first requirement is met.

By induction hyphotheses we have $I(\alpha_j) \wedge B \Rightarrow \alpha_j$ and $I(\alpha_k) \wedge B \Rightarrow \alpha_k$. By Lemma 5.2.1 we have $\alpha_j \wedge \alpha_k \wedge B \Rightarrow \alpha_i$. Hence, $I(\alpha_j) \wedge I(\alpha_k) \wedge B \Rightarrow \alpha_j \wedge \alpha_k \wedge B \Rightarrow \alpha_i$. Note that $v \notin B$, since v is A-local and $v \notin \alpha_i$, since α_i was created by resolution over v. Hence, by Lemma 5.2.6 $VE(I(\alpha_j) \wedge I(\alpha_k), v) \wedge B \Rightarrow \alpha_i$ and the second requirement is met.

Finally, by construction $I(\alpha_i)$ contains the union of all the variables in $I(\alpha_j)$ and $I(\alpha_k)$ with the exception of v. Also by construction, the A-local variables of α_i comprise the union of all the A-local variables of α_j and α_k with the exception of v. These facts and the induction hypothesis for the third requirement for α_j and α_k yield that the third requirement for α_i is met.

□

Theorem 5.2.8. *Given an A-resolution refutation, IGCI returns an interpolant.*

Proof. The set $I(\square)$ is the clause-interpolant for $\square$ by Lemma 5.2.7, hence it is an interpolant by Lemma 5.2.4. □

Unfortunately, in practice IGCI is still not always scalable because of the exponential blow-up resulting from variable elimination.

Now we are ready to present a scalable algorithm for the construction of an approximated interpolant by generating a B_{weak}-interpolant. The first stage of our algorithm traverses the resolution refutation to generate a non-global-interpolant. The second stage uses bounded variable elimination and then incomplete variable elimination (defined below), if required, to convert the non-global-interpolant to a B_{weak}-interpolant.

Definition 5.2.9 (Incomplete Variable Elimination)**.** Given a CNF formula F and a variable $v \in \text{Vars}(F)$, *incomplete variable elimination* is an operation that removes v from F by replacing clauses containing the variable v with the set $IVE(F, v)$ which contains *some* of the results of a pairwise resolution between F_v^+ and F_v^-, where two requirements are met:

1. $|IVE(F, v)| \leq |F_v^+| + |F_v^-|$
2. Let $\alpha \in F_v^+$ (F_v^-) be a clause, if there exists a clause $\beta \in F_v^-$ (F_v^+), such that $\alpha \otimes^v \beta$ is not a tautology, then there exists a clause $\gamma \in F_v^-$ (F_v^+) where $\alpha \otimes^v \gamma \in IVE(F, v)$ and $\alpha \otimes^v \gamma$ is not a tautology.

The idea behind incomplete variable elimination is to omit some of the resolvents when eliminating the variable v in order not to increase the number of clauses, yet to guarantee that each clause containing v has some contribution to the generated set of clauses. Note that while incomplete variable elimination is not sufficient to maintain unsatisfiability for all cases, it may

be sufficient for some cases. Incomplete variable elimination is not uniquely defined.

Our implementation of the procedure is provided in Figure 5.4.

Before presenting our final algorithm, we need to introduce the notion of a non-global-clause-interpolant:

Definition 5.2.10 (Non-Global-Clause-Interpolant)**.** Let (A, B) be an unsatisfiable pair of CNF formulas. Let α be a clause. Then, $I(\alpha)$ is a *Non-Global-Clause-Interpolant* of α iff:

- $A \Rightarrow I(\alpha)$
- $I(\alpha) \wedge B \Rightarrow \alpha$

Note that a non-global-clause-interpolant of $\square$ is a non-global-interpolant.

Consider now the algorithm described in Figure 5.5. Its first part (lines 2-21) traverses the resolution refutation and asssociates a non-global-clause-interpolant with each clause. Consider a visited clause $\alpha_i = \alpha_j \otimes^v \alpha_k$ when v is local. First, the algorithm sets $I(\alpha_i)$ to be the union of $I(\alpha_j)$ and $I(\alpha_k)$. It eliminates the variable v if the following two conditions hold: First, that eliminating v does not increase the clause size of $I(\alpha_i)$ (as in the bounded variable elimination of SatELite [27]), and second, that variable elimination has been performed for all clauses backward reachable from α_i. (The second condition is ensured by using an auxiliary set *Skipped* for marking clauses for which variable elimination was skipped). The second stage of the algorithm (starting from line 22) uses bounded variable elimination and then incomplete variable elimination to convert the non-global-interpolant to the eventually returned B_{weak}-interpolant by eliminating A-local variables. Note that the bounded variable elimination stage is non-redundant even though bounded variable elimination was performed locally for resolution refutation clauses, since sometimes bounded variable elimination is possible given a large set of clauses while it is impossible given a subset of that set. Note also that the algorithm returns an interpolant rather than merely a B_{weak}-interpolant if

1: **function** IVE(F,v)
2: $IVE(F, v) := \{\}$
3: **for all** $\alpha \in F_v^+$ **do**
4: **if** There exists a non-marked clause $\beta \in F_v^-$, such that $\alpha \otimes^v \beta$ is not a tautology **then**
5: $IVE(F, v) := IVE(F, v) \cup \alpha \otimes^v \beta$ // β: the first clause that meets if-condition
6: Mark β
7: **else**
8: **if** There exists a clause $\beta \in F_v^-$, such that $\alpha \otimes^v \beta$ is not a tautology **then**
9: $IVE(F, v) := IVE(F, v) \cup \alpha \otimes^v \beta$ // β: the first clause that meets if-condition
10: **end if**
11: **end if**
12: **end for**
13: **for all** Unmarked $\beta \in F_v^-$ **do**
14: **if** There exists a clause $\alpha \in F_v^-$, such that $\alpha \otimes^v \beta$ is not a tautology **then**
15: $IVE(F, v) := IVE(F, v) \cup \alpha \otimes^v \beta$ // β: the first clause that meets if-condition
16: **end if**
17: **end for**
18: **return** $IVE(F, v)$
19: **end function**

Figure 5.4: Incomplete Variable Elimination

all the A-local variables are succesfully removed before incomplete variable elimination is applied.

Next, we present the proof of Algorithm 5.5 starting with a number of lemmas.

Lemma 5.2.11. *A non-global-clause-interpolant of the empty clause $\square$ is a non-global-interpolant.*

Proof. Each of the two conditions in the definition of a non-global-clause-interpolant yields the corresponding condition in the definition of the non-global-interpolant. $\square$

Lemma 5.2.12. *Each iteration of both for-loops of Algorithm 5.5 constructs a non-global-clause-interpolant $I(\alpha_i)$ for the currently traversed clause α_i.*

Proof. The proof is only briefly sketched here, since it is a simpler version of the proof of Lemma 5.2.7. The proof is again by induction on the number of loop iterations. One needs to prove only the first two clause-interpolant properties in this lemma as compared to the three properties which need to be proved for Lemma 5.2.7 (which makes the proof simpler). Another difference is that variable elimination is not always applied when $\alpha_i = \alpha_j \otimes^v \alpha_k$, where v is A-local. However, the proof for both properties is straightforward when variable elimination is not applied. $\square$

Theorem 5.2.13. *Given an A-resolution refutation, Algorithm 5.5 returns a B_{weak}-interpolant.*

Proof. At line 22 of the algorithm, the set $I(\square)$ is the non-global-clause-interpolant for $\square$ by Lemma 5.2.12, hence it is a non-global-interpolant by Lemma 5.2.11. It is not hard to see that after bounded variable elimination is applied, $I(\square)$ is still a non-global-interpolant (or an interpolant if all the A-local variables are eliminated). If incomplete variable elimination is applied, it clearly maintains the property $A \Rightarrow I(\square)$, while at the end of the procedure the property $\mathrm{Vars}(I) \subseteq \mathrm{Vars}(A) \cap \mathrm{Vars}(B)$ is met, since all the A-local variables are eliminated. Hence $I(\square)$ is a B_{weak}-interpolant. $\square$

```
1: function SIG(π_A = (α_1, α_2, ..., α_q, α_{q+1}, α_{q+2}, ..., α_p ≡ □))
2:     Skipped := {}
3:     for all i ∈ {1, 2, ..., q} do
4:         I(α_i) := {α_i}
5:     end for
6:     for all i ∈ {q + 1, q + 2, ..., p ≡ □} do
7:         if α_i has exactly one parent β then
8:             I(α_i) := I(β)
9:         else
10:            if α_i = α_j ⊗^v α_k, where v is global then
11:                I(α_i) := I(α_j) ∪ I(α_k)
12:            else      // α_i = α_j ⊗^v α_k, where v is A-local
13:                I(α_i) := I(α_j) ∪ I(α_k)
14:                if |VE(I(α_j) ∪ I(α_k), v)| ≤ |I(α_j) ∪ I(α_k)| and {α_j, α_k} ∩
   Skipped = ∅ then
15:                    I(α_i) := VE(I(α_i), v)
16:                else
17:                    Skipped := Skipped ∪ {α_i}
18:                end if
19:            end if
20:        end if
21:    end for
22:    Apply bounded variable elimination for A-local variables over I(□)
23:    if I(□) then do not contain A-local variables
24:        return I(□)       // In this case I(□) is an interpolant
25:    else
26:        Apply incomplete variable elimination for A-local variables over
   I(□)
27:        return I(□)       // In this case I(□) is a B_weak-interpolant
28:    end if
29: end function
```

Figure 5.5: B_{weak}-Interpolant Generation

5.3 Using B_{weak}-Interpolants In Model Checking

In this section we describe a model checking algorithm that uses B_{weak}-interpolants. Our algorithm is composed of two main stages. Recall that by Definition 5.1.2, a B_{weak}-interpolant fulfills two out of the three conditions of an interpolant. Therefore, the first stage transforms the B_{weak}-interpolant into an interpolant.

The second stage uses interpolants computed by the first stage. In essence, the second stage is a modification of the original ITP and is called CNF-ITP. Besides the fact that CNF-ITP uses interpolants in CNF, it further takes advantage of this fact by applying optimizations which are possible only as a result of using interpolants in CNF.

Before going into the details of CNF-ITP, we describe ITP.

5.3.1 Interpolation-Based Model Checking Revisited

We have described ITP in detail in Chapter 2, we give a quick overview here. ITP [43] is a complete SAT-based model checking algorithm. It uses interpolation to over-approximate the reachable states in a transition system M with respect to a property p. ITP uses nested loops where the outer loop increases the depth of unrolling and the inner loop computes the reachable states. ITP is described in Figure 5.6

Definition 5.3.1. Let k and n be the depth of unrolling used in the outer loop and the iteration of the inner loop of ITP, respectively. Let us denote $\bar{F} = \langle INIT, I_1^k \ldots, I_n^k \rangle$ the resulting FRS. We define $R_n^k = INIT \vee I_1^k \vee I_2^k \vee \ldots \vee I_n^k$ to be the set of reachable states computed by the inner loop of ITP after n iterations and with respect to unrolling depth k. For a given $1 \leq j \leq n$, I_j^k is the interpolant computed in the j-th iteration of the inner loop.

From this point and on, k and n refer to the depth of unrolling used in

```
1: function ITP(M,p)
2:     if INIT ∧ ¬p == SAT then
3:         return cex
4:     end if
5:     k = 1
6:     while true do
7:         result = ComputeReachable(M, p, k)
8:         if result == fixpoint then
9:             return Valid
10:        else if result == cex then
11:            return cex
12:        end if
13:        k = k + 1
14:    end while
15: end function
```

Figure 5.6: Interpolation-Based Model Checking (ITP)

the outer loop and the iteration of the inner loop of ITP, respectively.

In general, the inner loop checks a fixed-bound BMC [4] formula where at each iteration only the initial states are replaced with an interpolant computed at a previous iteration (lines 22 and 30). This is done until the BMC formula becomes SAT (line: 30) or until a fixpoint is reached (lines: 25-27). In the former case, the outer loop increases the unrolling depth by 1[2] (line: 13) in order to either increase the precision of the over-approximations or to find a counterexample.

Lemma 5.3.2. *$R_n^k(V^0) \wedge path^{0,k-1} \wedge (\bigvee_{j=0}^{k-1} \neg p(V^j))$ is unsatisfiable.*

Proof. The proof is immediate from the interpolant definition (Definition 2.3.1) and from the definition of R_n^k. Let k and n be the depth of unrolling used in the outer loop and the iteration of the inner loop of ITP respectively and let $\bar{F} = \langle I_0^k = INIT, I_1^k \ldots, I_n^k \rangle$ be the FRS computed by ITP. For

[2]Some works choose different ways of increasing k. For example, k can be increased by the number of iterations executed in the inner loop: $k = k + n$. In our experiments $k = k + 1$ yielded better results.

16: **function** ComputeReachable(M,p, k)
17: $R_0^k = INIT$, $I_0^k = INIT$, $n = 1$
18: **if** $I_0^k \wedge \text{path}^{0,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k)) == \text{SAT}$ **then**
19: **return** *cex*
20: **end if**
21: **repeat**
22: $A = I_{n-1}^k(V^0) \wedge TR(V^0, V^1)$
23: $B = \text{path}^{1,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k))$
24: $I_n^k = \text{GetInterpolant}(A, B)$
25: **if** $I_n^k \Rightarrow R_{n-1}^k$ **then**
26: **return** $fixpoint$
27: **end if**
28: $R_n^k = R_{n-1}^k \vee I_n^k$
29: $n = n + 1$
30: **until** $I_{n-1}^k \wedge \text{path}^{0,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k)) == \text{SAT}$
31: **end function**

Figure 5.7: Inner loop of ITP

$0 \leq j \leq n-1$, $I_j^k(V^0) \wedge \text{path}^{0,k} \wedge (\bigvee_{j=0}^{k} \neg p(V^j))$ is unsatisfiable and I_{j+1}^k is the interpolant derived from the proof of unsatisfiability for this formula. Thus, for $0 \leq j \leq n$ $I_j^k(V^0) \wedge \text{path}^{0,k-1} \wedge (\bigvee_{j=0}^{k-1} \neg p(V^j))$ is unsatisfiable, and by that $R_n^k(V^0) \wedge \text{path}^{0,k-1} \wedge (\bigvee_{j=0}^{k-1} \neg p(V^j))$ is unsatisfiable. □ □

R_n^k is also referred to as $(k-1)$-adequate.

5.3.2 Transforming a B_{weak}-Interpolant Into an Interpolant Using Inductive Reasoning

As we have shown in Section 5.2, given a pair of formulas (A, B) such that $A \wedge B$ is unsatisfiable, a B_{weak}-interpolant I_w can be computed. By Definition 5.1.2, $A \Rightarrow I_w$ and $\text{Vars}(I_w) \subseteq \text{Vars}(A) \cap \text{Vars}(B)$, but it is not guaranteed that $I_w \wedge B$ is unsatisfiable. Intuitively, we can think of I_w as being too over-approximated and therefore needing strengthening with respect to B.

Definition 5.3.3 (B-adequate)**.** Let (A, B) be a pair of formulas s.t. $A \wedge B \equiv \bot$ and let I_w be a B_{weak}-interpolant for (A, B). We say that I_w is *B-adequate* iff $I_w \wedge B \equiv \bot$.

Following the above definition, our purpose is to make a B_{weak}-interpolant I_w B-adequate. We refer to this procedure as *B-Strengthening*.

The purpose of this section is to demonstrate the use of B_{weak}-interpolants for model checking, in particular in the context of ITP.

Definition 5.3.4 (k-n-pair)**.** Given the formulas $A = I^k_{n-1}(V^0) \wedge TR(V^0, V^1)$ and $B = \text{path}^{1,k} \wedge (\bigvee_{i=1}^{k} \neg p(V^i))$. The pair (A, B) is called a *k-n-pair*. When $A \wedge B \equiv \bot$ we call (A, B) an *inconsistent k-n-pair*.

Consider a run of ITP for a given k and n. We aim at computing I^k_n. Let (A, B) be an inconsistent k-n-pair and let I_w be the B_{weak}-interpolant for (A, B). If I_w is B-adequate then it is an interpolant and therefore I^k_n can be defined to be I_w. If I_w is not B-adequate we are required to apply B-Strengthening and transform I_w into an interpolant.

Let us assume that I_w is not B-adequate and that $I_w(V^1) \wedge B$ is satisfiable. There exists a state $s \in I_w$ such that $s(V^1) \wedge B$ is satisfiable. Intuitively, in order to make I_w B-adequate, and by that an interpolant, we would like to remove s from it.

Clearly, $A \wedge s(V^1)$ is unsatisfiable; otherwise $A \wedge B$ would have been satisfiable. Thus, B-Strengthening can be done by iterating all assignments for $I_w(V^1) \wedge B$, extracting a state $s \in I_w$ from an assignment and blocking it in I_w. This is an inefficient way to perform B-Strengthening since the number of such assignments may be too large.

To overcome this, we use knowledge about the problem at hand. Namely, we take into account the fact that A is of the following form: $A = I^k_{n-1}(V) \wedge TR(V, V')$.

Definition 5.3.5 (Relatively Inductive)**.** Let R and Q be propositional formulas and M a transition system. We say that Q is *relatively inductive* with

respect to R and M if $(R(V) \wedge Q(V)) \wedge TR(V, V') \Rightarrow Q(V')$. When M is clear from the context we omit it.

Recall that by Definition 5.3.1 R_n^k represents an over-approximation of all reachable states after up to n transitions and it is $(k-1)$-adequate (Lemma 5.3.2).

Lemma 5.3.6. *Let (A, B) be an inconsistent k-n-pair. Let I_w be the B_{weak}-interpolant for (A, B) . If s is an assignment to V s.t. $s(V^1) \wedge B$ is satisfiable, then the following holds:*

- $R_{n-1}^k \Rightarrow \neg s$
- $R_{n-1}^k \wedge \mathrm{TR} \Rightarrow \neg s'$

Proof. From Lemma 5.3.2 we know that $R_{n-1}^k(V^0) \wedge \mathrm{path}^{0,k-1} \wedge (\bigvee_{i=0}^{k-1} \neg p(V^i))$ is unsatisfiable. Since s represents a state that can reach the bad states in $k-1$ steps or less, s cannot be part of R_{n-1}^k. Thus, $s \notin R_{n-1}^k$ and $R_{n-1}^k \Rightarrow \neg s$. We now need to show that s cannot be reached from R_{n-1}^k. By Definition 5.3.1, $R_{n-1}^k = INIT \vee I_1^k \vee \ldots \vee I_{n-1}^k$. Since (A, B) is an inconsistent k-n-pair, $I_{n-1}^k(V^0) \wedge TR(V^0, V^1) \wedge \mathrm{path}^{1,k} \wedge (\bigvee_{i=1}^{k} \neg p(V^i))$ is unsatisfiable. By that, and by the fact that $R_{n-1}^k(V^0) \wedge \mathrm{path}^{0,k-1} \wedge (\bigvee_{i=0}^{k-1} \neg p(V^i))$ is unsatisfaible (R_{n-1}^k is $(k-1)$-adequte) we know that $R_{n-1}^k(V^0) \wedge \mathrm{path}^{0,k} \wedge (\bigvee_{i=0}^{k} \neg p(V^i))$ is unsatisfiable. Using the same reasoning as before, since s represents a state that can reach the bad states in $k-1$ steps or less, it cannot be reached from R_{n-1}^k and thus it is immediate that $R_{n-1}^k \wedge TR \Rightarrow \neg s'$. □ □

The above lemma states that if a state s can reach a bad state in up to $k-1$ transitions, it cannot be a state in the set R_{n-1}^k. Consider a B_{weak}-interpolant I_w derived from the pair (A, B), and assume $s \in I_w$ (derived from the satisfying assignment to $I_w(V^1) \wedge B$), then s follows the condition in Lemma 5.3.6. Therefore, $R_{n-1}^k \Rightarrow \neg s$ and $R_{n-1}^k \wedge TR \Rightarrow \neg s'$ hold and by that $(R_{n-1}^k \wedge \neg s) \wedge TR \Rightarrow \neg s$ holds. By Definition 5.3.5 $\neg s$ is relatively inductive with respect to R_{n-1}^k. Therefore, $\neg s$ can be inductively generalized [8].

32: **function** FINDMISSINGCLAUSES(R,I_w, B, n)
33: $C = \emptyset$
34: **while** $(I_w \wedge C)(V^1) \wedge B == \text{SAT}$ **do** // When $C = \emptyset$ it is evaluated as $\top$
35: Get $s \in I_w$ from the SAT assignment
36: $c = \text{INDUCTIVEGENERALIZATION}(R, s, C)$
37: $C = C \cup c$
38: **end while**
39: STORECLAUSES(n)
40: **return** C
41: **end function**

Figure 5.8: Find the clauses needed for the B_{weak}-interpolant I_w to be B-adequate

Inductive generalization results in a sub-clause c of $\neg s$ such that $(R^k_{n-1} \wedge c) \wedge TR \Rightarrow c'$ and $INIT \Rightarrow c$. c can then be used to strengthen I_w and R^k_{n-1}. Adding the clause c to I_w removes s from I_w. This process is then iterated until I_w becomes B-adequate and hence an interpolant. The algorithm for finding the clauses that make I_w B-adequate is described in Figure 5.8.

Theorem 5.3.7. *Let (A, B) be an inconsistent k-n-pair. Let I_w be a B_{weak}-interpolant and let $c_1, \ldots, c_m$ be clauses s.t.* INIT $\Rightarrow c_i$ *and c_i is relatively inductive with respect to R^k_{n-1} for $1 \leq i \leq m$. If $(I_w \wedge \bigwedge_{j=1}^m c_j) \wedge B \equiv \bot$ then $I_w \wedge \bigwedge_{i=1}^m c_i$ is an interpolant w.r.t $A = (I^k_{n-1}(V^0) \wedge \bigwedge_{j=1}^m c_j(V^0)) \wedge \text{TR}(V^0, V^1)$ and $B = path^{1,k} \wedge (\bigvee_{i=1}^k \neg p(V^i))$.*

Proof. Let us denote $I = I_w \wedge \bigwedge_{i=1}^m c_i$. Clearly, $I \wedge B$ is unsatisfiable and I is over the shared variables of A and B. We only need to show that $A \Rightarrow I$. Since I_w is a B_{weak}-interpolant, $A \Rightarrow I_w$. Now we only need to show that $A \Rightarrow \bigwedge_{i=1}^m c_i$. Each c_i is relatively inductive w.r.t. R^k_{n-1}. In particular, $I^k_{n-1}(V^0) \wedge TR(V^0, V^1) \Rightarrow c_i$ for $1 \leq i \leq m$. Thus, $A \Rightarrow c_i$ for $1 \leq i \leq m$ and therefore $A \Rightarrow \bigwedge_{i=1}^m c_i$. Thus, $A \Rightarrow I$. □ □

5.3.3 CNF-ITP: Using B_{weak}-Interpolants in ITP

Above we described how a B_{weak}-interpolant is transformed into an interpolant efficiently for model checking. In this section we present CNF-ITP, a model checking algorithm that is based on ITP. CNF-ITP uses the method described above to compute interpolants. In addition, it uses optimizations that are possible as a result of using interpolants in CNF.

Like the original ITP, our version consists of two nested loops. Since the computation of interpolants is performed in the inner loop, this is where we have made most of our modifications and optimizations. Recall that in the inner loop a BMC formula of a fixed-bound is checked iteratively, where at each iteration only the initial states are replaced by the interpolants computed in the previous iteration. Our modified version of the inner loop appears in Figure 5.9

As before, we consider k to be the unrolling depth set by the outer loop and used in the inner loop and n to be the iteration during the execution of the inner loop.

The beginning of the loop is similar to the original inner loop of ITP. First, a counterexample of length k is checked (lines: 44-46). If no counterexample exists the pair (A, B) is defined and a B_{weak}-interpolant I_w is computed (line: 50). Then, two optimizations are applied. First, clauses are pushed forward (line: 51). Second, previously computed interpolant is conjoined to the currently computed B_{weak}-interpolant (line: 52). We will go into more details in the next section. Since I_w may not be B-adequate, the B-Strengthening process may need to add clauses to it (to strengthen it). Adding clauses to I_w before B-Strengthening results in a more efficient B-Strengthening. Moreover, after pushing clauses forward and adding clauses from the previously computed interpolant, I_w may become B-adequate, thereby rendering B-Strengthening redundant.

After applying the two optimizations, B-Strengthening is invoked (line 53). Then the clauses learned during this process are conjoined with R^k_{n-1} and

I^k_{n-1} (line 56), and I_w (line 57). After conjoining the clauses, I^k_n is an interpolant. The rest of the loop is identical to the original inner loop of ITP.

We now describe the optimizations in more detail.

5.3.3.1 Pushing Clauses Forward

Let us consider the interpolant I^k_n computed during the n-th iteration of the inner loop. Since I^k_n is given in CNF, assume that $I^k_{n-1} = \{c_1, \ldots, c_m\}$ where c_i is a clause for every $1 \leq i \leq m$.

Definition 5.3.8. Let M be a transition system and let $F = \{c_1, \ldots, c_m\}$ be a formula in CNF where c_i is a clause over V for every $1 \leq i \leq m$. A clause c_i for some $1 \leq i \leq n$ is said to be *pushable* if $F(V) \wedge TR(V, V') \Rightarrow c_i(V')$ holds.

After the computation of a B_{weak}-interpolant I_w, we try to find pushable clauses in the previous interpolant. Those clauses can be made part of the new interpolant. More precisely, if a clause $c_i \in I^k_{n-1}$ is pushable then we add it to I_w such that $I_w = I_w \wedge c_i$. Adding the pushable clauses to the B_{weak}-interpolant strengthens it and may make it B-adequate.

5.3.3.2 Incremental Interpolants

The outer loop of CNF-ITP (and ITP) increases the unrolling depth when a more precise over-approximation is needed. Let I^1_1 be the interpolant computed in the first iteration of the inner loop for $k = 1$ and let I^2_1 be the interpolant computed in the first iteration of the inner loop for $k = 2$. Clearly, since both I^1_1 and I^2_1 over-approximate the states reachable in one transition from the initial states, $I^1_1 \wedge I^2_1$ is also an over-approximation of the same set of states. Usually, the size of the interpolants is an issue. Therefore, whenever the inner loop terminates and the bound is increased, all computed interpolants are discarded and are not re-used [42]. Since our method produces interpolants in CNF that are usually small, this conjunction does not create

42: **function** ComputeReachableCNF(M,p, k)
43: $R_0^k = INIT$, $I_0^k = INIT$, $n = 1$
44: **if** $I_0^k \wedge \text{path}^{0,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k)) ==$ SAT **then**
45: **return** cex
46: **end if**
47: **repeat**
48: $A = I_{n-1}^k(V^0) \wedge TR(V^0, V^1)$
49: $B = \text{path}^{1,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k))$
50: $I_w =$ GetBWeakInterpolant(A, B)
51: PushInductiveClauses$(I_w, n-1)$
52: $I_w = I_w \wedge I_n^{k-1}$ // For $k = 1$, $I_n^0 = \top$
53: $C =$ FindMissingClauses(R_{n-1}^k, I_w, B)
54: $I_n^k = I_w$
55: **for all** $c \in C$ **do**
56: $R_{n-1}^k = R_{n-1}^k \wedge c$ // Implicitly conjoining c with I_{n-1}^k
57: $I_n^k = I_n^k \wedge c$
58: **end for**
59: **if** $I_n^k \Rightarrow R_{n-1}^k$ **then**
60: **return** $fixpoint$
61: **end if**
62: $R_n^k = R_{n-1}^k \vee I_n^k$
63: $n = n + 1$
64: **until** $I_{n-1}^k \wedge \text{path}^{0,k} \wedge (\neg p(V^1) \vee \ldots \vee \neg p(V^k)) ==$ SAT
65: **end function**

Figure 5.9: Inner loop of CNF-ITP

huge CNF formula. This re-use of previously computed interpolants increases the efficiency of CNF-ITP as compared to ITP.

5.4 Experimental Results

Our approach includes two major parts. The first part computes a B_{weak}-interpolant from a resolution refutation, and the second part applies B-Strengthening and a model checking algorithm CNF-ITP. The computation

of B_{weak}-interpolants was implemented on top of *MiniSAT 2.2*. CNF-ITP and ITP were implemented in a closed-source model checker. For IC3 we used the publicly available ABC framework[3]. In the results we also include the runtime for ABC's ITP implementation in order to show the efficiency of our implementation.

To evaluate our method we used a representative subset of the HWMCC'12 benchmark set. All experiments were conducted on a system with an Intel E5-2687W running at 3.1GHz with 32GB of memory. Timeout was set to 900 seconds. As mentioned, we sought to test two aspects: the size of the resulting interpolants and the impact on model checking.

In Figure 5.10 a comparison between interpolants sizes is presented. Note that in the majority of cases, interpolants generated by CNF-ITP are orders of magnitude smaller than those generated by the traditional method (For ITP, the number of clauses is after translation of the interpolants to CNF). Considering the entire set of benchmarks we see that CNF-ITP generates interpolants that are 117 times smaller than those generated by the traditional method.

Comparing the run-time of the model checking algorithms shows that our CNF-ITP algorithm outperforms ITP and IC3 in terms of the overall run-time on this subset. CNF-ITP outperforms ITP on 32 instances, where in 16 of these instances ITP times out. ITP outperforms CNF-ITP in 21 cases only. CNF-ITP outperforms IC3 in 18 cases, but IC3 is preferable in 23 cases. CNF-ITP is the absolutely best algorithm in 14 cases. Figure 5.11 shows a comparison of CNF-ITP to the other two algorithms.

Table 5.1 presents a detailed analysis of the experiments. We chose all valid benchmarks that either ITP or CNF-ITP could prove in the given time frame (55 cases). Consider Table 5.1. As was shown in Figure 5.10 our method generates significantly smaller interpolants in almost every case. Summarizing the average size of all computed interpolants shows that CNF-

[3]https://bitbucket.org/alanmi/abc

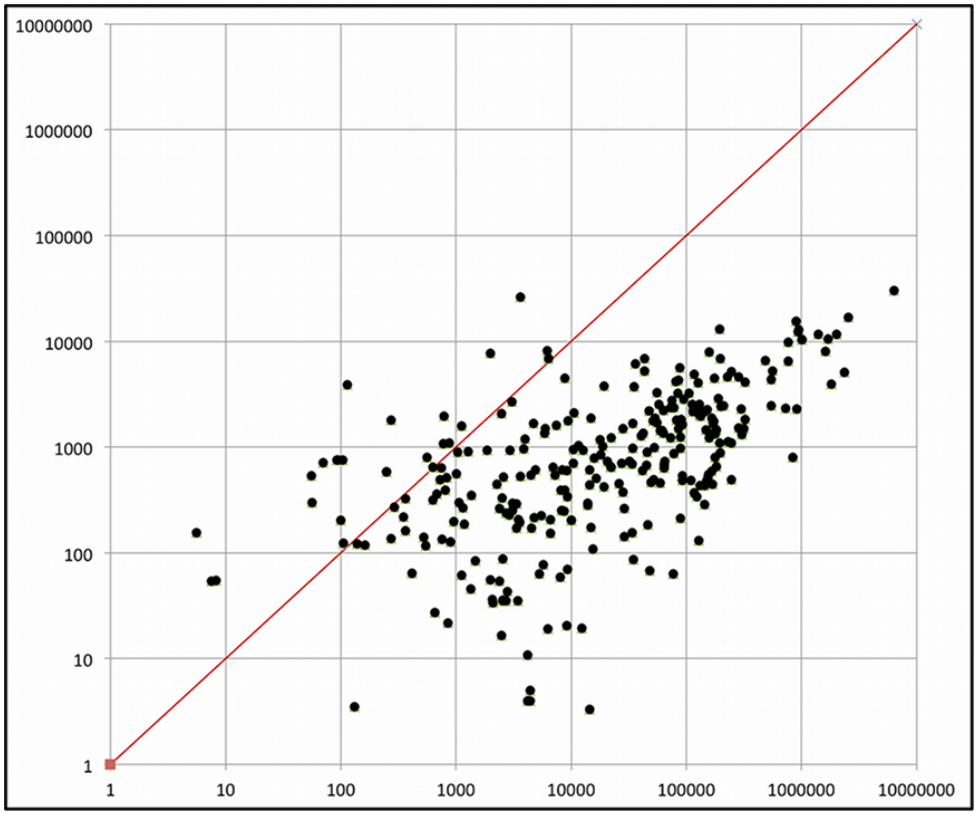


Figure 5.10: Comparing sizes of generated interpolants. Y-axis represents interpolants generated by CNF-ITP and X-axis represents interpolants generated by ITP.

ITP generates interpolants that are *49 times smaller* than those generated by ITP. Note that average interpolant computation time is in the same ballpark for both methods.

Another interesting aspect of the comparison between CNF-ITP and ITP is the convergence bound. We can see that in many cases the bound is different. This indicates that the strength of the interpolants computed by the two methods is different and affects the results of the model checking algorithm.

Analysis of the results in the table shows that whenever the number of clauses in the interpolants computed by CNF-ITP is significantly smaller than the number of clauses in the interpolants computed by ITP, the former performs better.

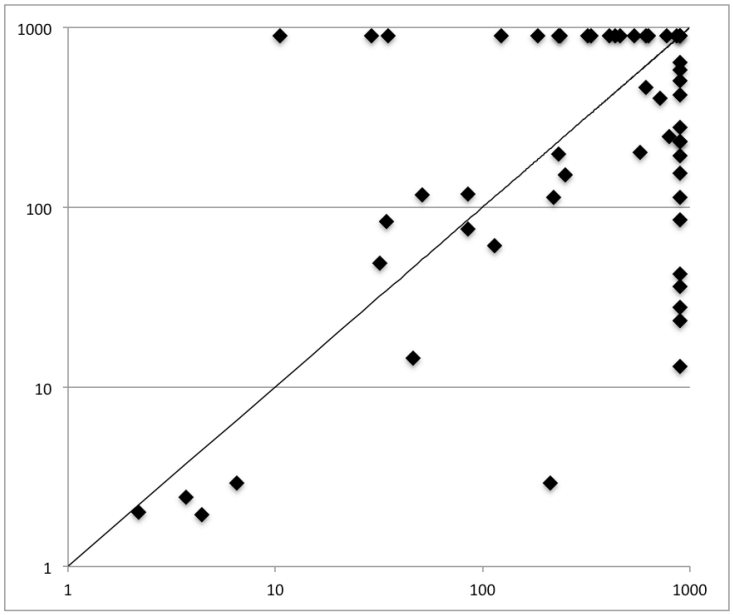
In the cases where the size of interpolants is fairly the same, ITP performs better. This can be explained by the fact that ITP computes small interpolants when the resolution refutation is small. Therefore, computing the interpolants in ITP is more efficient in these cases since it only requires linear traversal over the resolution refutation. In contrast, our method requires B-Strengthening, a process that is in some cases expensive. We conclude that when the resulting interpolants in ITP are large, CNF-ITP has a significant advantage in the vast majority of cases.

When analyzing results from the entire HWMCC'12 benchmark, we have found that in CNF-ITP 95% of the clauses are generated using SIG. This result shows the importance of SIG when computing the interpolants. Yet, B-strengthening is also critical, since only around 25% of the instances were solved solely by SIG.

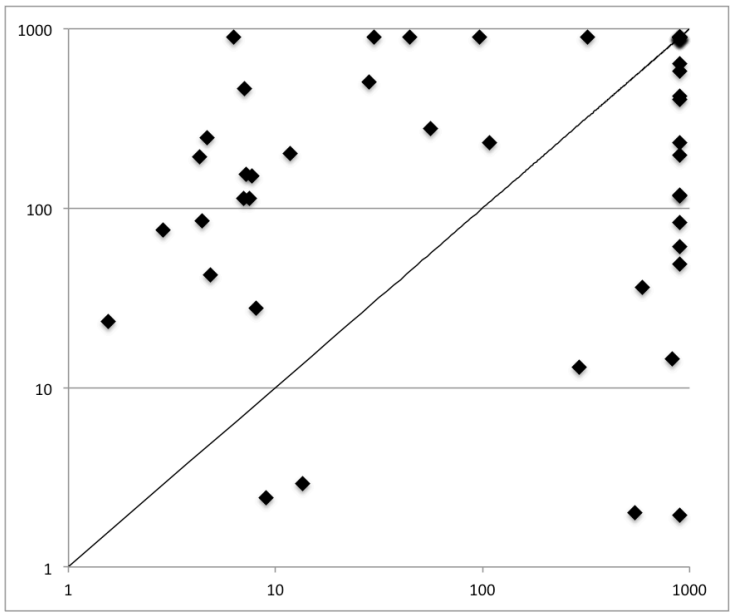
5.5 Conclusions

CNF-ITP uses key elements of ITP and IC3. On the one hand, like ITP, CNF-ITP uses the resolution refutation to get information about the reach-

able states. This information is only partial, and therefore CNF-ITP also uses *inductive generalization*, a key element of IC3, to complete the computation of reachable states. Since the reachable states are computed by means of over-approximations, there are cases in which the precision of these approximations must be increased. To do so, CNF-ITP uses unrolling, like in ITP. In addition, it uses the fact that interpolants are given in CNF and tries to reuse clauses that have already been learnt (both by pushing the clauses forward and by using previously computed interpolants). CNF-ITP can be viewed as a hybridization of the monolithic approach (ITP) and the incremental approach (IC3). We believe that there are well-founded grounds for comparing the three algorithms, and that further development can bring about an even tighter integration of ITP and IC3. This discussion, however, is outside the scope of this paper and is an avenue for future research.



(a) Runtime ITP vs. CNF-ITP.



(b) Runtime IC3 vs. CNF-ITP

Figure 5.11: Y-axis represents CNF-ITP's runtime in seconds. X-axis represents runtime in seconds for the compared algorithm (IC3 or ITP). Points below the diagonal are in favor of CNF-ITP.

Table 5.1: Experiment parameters on part of the benchmarks. *Name*: property name; ♯*Vars*: number of state variables in the cone of influence; k is the bound of the outer loop at which fixpoint was found; total_n is the *total* number of iterations executed by the inner loop; $\text{clauses}_{\text{Avg}}$ is the average number of clauses representing each computed interpolant; Extract[s] is the average time to compute an interpolant in seconds; MC[s] is the total runtime of the algorithm in seconds. Values in boldface are the best of all three. Underlined runtime is for cases where CNF-ITP outperforms ITP and *Italic* is for cases where CNF-ITP outperforms IC3.

		IC3_{ABC}	ITP_{ABC}	ITP					CNF-ITP				
Name	♯Vars	MC[s]	MC[s]	k	total_n	$\text{clauses}_{\text{Avg}}$	Extract[s]	MC[s]	k	total_n	$\text{clauses}_{\text{Avg}}$	Extract[s]	MC[s]
beembkry1b1	76	**4.68**	758	15	72	94495	3.14	792	20	83	**1830**	0.65	248
beembrptwo1f2	227	**7.02**	TO	59	650	2905	0.83	TO	40	1135	**227**	0.03	114
beembrptwo5f2	227	292	TO	81	1105	**5**	0.29	TO	29	1326	155	0.01	**13**
beemcoll1b1	132	**11.77**	TO	9	51	45563	2.09	577	11	52	**487**	0.28	201
beemexit5f1	246	**7.11**	106	25	218	15792	1.21	611	25	255	**792**	0.3	466
beemfish4f1	94	**4.41**	TO	15	50	63423	3.26	TO	14	59	**1348**	0.72	85
beemfwt1b1	1214	321.86	396	4	18	5721	0.23	**10.58**	4	15	**62**	38.14	TO
beemfwt5f2	3045	543.32	14.68	5	9	854	0.009	2.2	5	10	**22**	0.00	**2.01**
beemlmprt5f1	121	**4.29**	TO	14	65	59503	3.9	TO	12	72	**1423**	1.3	194
beemmsmie1b1	89	**28.2**	TO	4	12	230498	11	TO	5	18	**1126**	22	505
beemndhm2f2	251	13.53	138	7	49	29051	1.07	213	5	10	**143**	0.14	**2.92**
beempgmprot1b1	1025	**8.03**	97	33	218	1121	0.82	TO	19	113	**61**	0.00	27.65
beempgmprot7b1	1033	591.49	204	27	168	2717	1.23	TO	18	153	**237**	0.00	**36.21**
beemtlphn5f1	249	**29.81**	TO	12	60	73460	5.27	TO	20	66	**1197**	0.06	TO
beemrshr3b1	720	TO	TO	8	56	8043	2.17	437	8	41	**58**	16	TO
beemtrngt2b1	170	**1.55**	TO	29	193	31942	1.95	TO	15	154	**618**	0.08	23.42
beemtrngt4b1	228	**44.71**	TO	29	196	22144	1.56	TO	30	281	**371**	0.71	TO
bob05	2404	**7.5**	275	24	121	962	0.37	221	24	136	**198**	0.38	113
bob1u05cu	4377	**7.66**	235	24	124	3116	0.45	251	24	147	**272**	0.19	152
bobaesdinvdmit	1335	**56.12**	TO	4	12	122692	34.4	TO	3	21	**2408**	6.2	278
eijkbs3330	246	**7.2**	TO	3	6	764550	22.74	TO	3	9	**5873**	10.44	154
pdtpmsviper	580	**108**	TO	4	6	903732	13	TO	3	9	**2304**	21	233
6s38	1931	TO	TO	10	33	84988	1.19	TO	7	22	**4299**	15.7	**423**
6s102	1121	TO	TO	37	275	21947	0.5	TO	25	137	**654**	1.3	**233**
6s108	782	**4.83**	TO	8	43	89493	3.67	TO	7	25	**1787**	0.56	42.65
6s120	58	**0.71**	4.1	3	6	**365**	0.34	6.5	3	8	373	0.12	2.92
6s121	419	821.54	TO	24	214	4542	0.08	46.25	18	98	**389**	0.04	**14.42**
6s130	811	TO	**18.7**	7	17	43237	2.6	123	6	11	**5264**	55	TO
6s131	811	TO	**19.2**	9	21	75167	6	407	7	9	**2757**	42	TO
6s132	139	**2.87**	7.5	7	13	35973	2.88	85	5	13	**4221**	3.5	76
6s136	3342	TO	3.1	20	58	2471	0.005	4.4	20	53	**16**	0.00	**1.94**
6s151	150	TO	TO	14	515	**1998**	0.22	**461**	10	122	6226	3.15	TO
6s159	252	0.03	7.8	15	143	656	0.01	4.9	10	60	**27**	0.00	0.34
6s164	198	8.96	TO	18	77	753	0.02	3.7	18	85	**135**	0.006	**2.43**
6s181	607	TO	26.2	8	19	63509	4.58	**232**	6	10	**2556**	6.45	TO
6s183	607	TO	**18**	10	24	87782	1	612	7	9	**5641**	53.64	TO
6s189	2434	TO	TO	18	86	10103.779	1	**864**	15	29	**202**	0.12	TO
6s194	2389	TO	TO	47	300	8312.99	1	721	48	378	**606**	0.68	**403**
6s19	607	TO	**8.95**	8	19	78338.632	1	333	5	9	**2346**	7.72	TO
6s4	202	TO	TO	79	6758	56.597	1	TO	79	8633	**300**	0.01	**580**
6s51	3107	**96.65**	TO	52	416	3960.346	1	774	54	540	**1198**	1.045	TO
6s6	429	**6.29**	53.7	12	29	46353.31	1	540	12	29	**182**	2.677	TO
6s9	607	TO	29.2	8	19	63509	1	238	5	9	**2236**	6.901	TO
intel010	539	TO	TO	16	168	18092	0.4	**184**	17	354	**850**	2	TO
intel011	533	TO	TO	18	268	14526	0.4	**322**	20	544	**606**	1	TO
intel018	491	TO	TO	14	399	1186	0.02	**29**	16	491	**187**	1	TO
intel019	510	TO	TO	16	394	1345	0.02	**35**	16	388	**45**	1.5	TO
intel020	354	TO	TO	14	251	2388	0.04	**32**	15	379	**264**	0.06	49
intel021	365	TO	TO	18	316	2503	0.05	**51.3**	18	489	**331**	0.14	*117*
intel022	530	TO	TO	21	435	18818	0.5	**629**	20	555	**585**	1.05	TO
intel023	358	TO	TO	30	593	3339	0.1	233	32	1058	290	0.09	**197**
intel024	357	TO	TO	15	233	3087	0.04	**34.5**	15	341	**206**	0.12	*83*
intel028	7436	TO	TO	28	559	3564	0.2	TO	26	770	194	0.44	**642**
intel031	531	TO	TO	21	268	3465	0.15	114	18	235	**183**	0.03	**61**
intel034	3297	TO	TO	16	425	1477	0.02	**85**	16	432	**83**	0.19	*119*
Total		26442	33920			3279594	143	26519			66454	326	**22996**

Chapter 6

Lazy Abstraction and SAT-based Reachability

This work is based a paper that appeared in [58]. In this work we introduce a novel lazy abstraction-refinement technique for hardware model checking, integrated with the SAT-based algorithm IC3 [8].

Most SAT-based model checking algorithms are based on an unrolling of the model's transition relation in order to traverse its state space. In contrast, the recently introduced IC3 algorithm [8] avoids such unrolling. To verify a safety property, IC3 gradually builds a series of sets of states $F_0, \ldots, F_i, \ldots$, where F_i over-approximates the set of states reachable within i steps from the initial states. The computation moves back and forth along the F_i's and strengthens them by eliminating unreachable states. This is done via *local reachability checks* between consecutive sets F_i and F_{i+1}. IC3 either reaches a fixpoint, in which case all reachable states satisfy the desired property, or returns a counterexample.

Abstraction-refinement is a well known methodology for tackling the state-explosion problem. Abstraction hides model details that are not relevant for the checked property. The resulting abstract model is then smaller. *Lazy abstraction* [36, 44], developed for software model checking, is a spe-

cific type of abstraction that allows hiding different model details at different steps of the verification.

In this work we develop, for the first time, a *lazy* abstraction-refinement framework for hardware. We use the *visible variables abstraction* [40], which is particularly suitable for hardware. However, we use it in a lazy manner in the sense that different sets of visible variables are used in different iterations of the state-space traversal.

We find the IC3 algorithm most suitable for lazy abstraction since its state traversal is performed by means of local reachability checks, each involving only two consecutive sets. Thus, at each check a different set of variables is relevant.

Our model checking algorithm, called L-IC3, thus integrates a lazy abstraction-refinement mechanism into IC3. Similarly to IC3, L-IC3 computes a series of over-approximating sets F_i. However, L-IC3 considers abstractions of the model during this computation. When constructing F_{i+1}, we determine a set of variables U_i, needed for its construction, and abstract both *states and transitions* accordingly. The variables in U_i are referred to as "visible", while the others are invisible and treated as inputs.

The key ingredients of L-IC3 are therefore a series $\bar{F}$ of over-approximating sets of states F_i and an abstraction sequence $\bar{U}$ of sets of variables U_i.

L-IC3 works in stages. Each stage consists of an *abstract model checking* step, followed by a *refinement* step. At a given stage, the abstract model checking extends both $\bar{F}$ and $\bar{U}$ and checks if they include a potential abstract counterexample. If not, the sequences are further extended. If a potential abstract counterexample is found, the algorithm strengthens the sets F_i by eliminating abstract states that might be a part of an abstract counterexample.

We use a nonstandard notion of abstract counterexample, based on both $\bar{F}$ and $\bar{U}$. It consists of a sequence of abstract states connected by abstract transitions, satisfying:

1. each transition is based on a different abstraction U_i, and

2. each abstract state intersects the set F_i at the corresponding time frame

Our notion of counterexample reflects the incorporation of lazy abstraction into the mechanism of computing $\bar{F}$.

If an abstract counterexample is found, meaning that no strengthening is possible anymore based on the abstractions, the refinement step is invoked. Refinement applies just one iteration of a *concrete* variation of IC3, on the $\bar{F}$ computed by the abstract model checking. By doing so, it either finds a concrete counterexample or strengthens the F_i's so that *all* concrete counterexamples of length k are eliminated. In the latter case, the U_i's are also refined by adding more visible variables to each of them, *as* needed and *where* needed. Once refinement is finished we move to the next L-IC3 stage and the abstract model checking is re-invoked, continuing the computation from iteration k+1, with the refined sequences. This makes L-IC3 *incremental*.

L-IC3 terminates with either a fixpoint, in which case we conclude that the system satisfies the property, or with a concrete counterexample.

In summary, the main contribution of our work is a novel *lazy* abstraction-refinement technique for hardware. To the best of our knowledge this is the first time lazy abstraction is considered in the context of hardware. Our abstract model checking and refinement are SAT-based. Both avoid unrolling of the transition relation.

In order to evaluate our algorithm we compared it with IC3 on a set of large industrial designs and properties. We obtained speedups of up to two orders of magnitude. Our experiments demonstrate that our lazy abstraction indeed uses different sets of variables in different time frames. Moreover, only a small portion of the design's variables are used.

6.0.1 Related Work

[28] and [10] suggest optimizations and extensions to IC3, but they do not combine it with a lazy abstraction-refinement mechanism ([28] suggests the use of abstraction for IC3 but without implementation details nor results). In [45, 35, 32, 18, 33], SAT-based refinement is introduced. However, they use an unrolling of the model while we use local checks a-la IC3. Similarly to [45, 18], we also exploit an unSAT-core for refinement. However, we never unroll the model, while [45] does. Further, [45] is not incremental since after refinement it resumes its (abstract) model checking from time frame 0.

IC3 [8] is sometimes also viewed as an abstraction-refinement algorithm, since it refers to over-approximated sets F_i and the strengthening of these sets resembles refinement. However, the underlying model used by IC3 is concrete, and only the concrete transition relation is considered. We, on the other hand, alternate between abstract transition relations (in the abstract model checking step) and the concrete transition relation (in the refinement step). Our algorithm thus adds a layer of abstraction-refinement on top of this over-approximation-strengthening mechanism.

6.1 Preliminaries

In this chapter, we will use a slightly different version of a transition system than the one given in Defintion 2.0.1. In the following definition, we explicity distinguish state variables from input variables.

Definition 6.1.1. A *finite state transition system* (a model) is a tuple $M = (V, U, INIT, TR)$ where V is a set of Boolean variables, $U \subseteq V$ is a set of state variables, $V \setminus U$ is a set of input variables, $INIT(V)$ is a propositional formula over V describing the initial states, and $TR(V, V')$ describes a total transition relation which is defined as a propositional formula over V and the next-state variables $V' = \{v' \mid v \in V\}$.

The transition relation is described using next-state functions for each state variable. Namely, $TR(V,V') = \bigwedge_{v \in U} (v' = f_v(V,V'))$ where $f_v(V,V')$ is a propositional formula that assigns the next value to $v \in U$ based on current and next-state variables. Note that for an input variable $v \in V \setminus U$, f_v is not defined.

Definition 6.1.2. Let $\bar{F}$ be an MFRS (recall Definition 2.4.1). A formula η is *inductive* up to j, if $F_j \wedge \eta \wedge TR \Rightarrow \eta'$. η is an *invariant* up to level j if $F_i \Rightarrow \eta$ holds for each $i \leq j$.

Note that if η is inductive up to j then $F_i \wedge \eta \wedge TR \Rightarrow \eta'$ holds for each $i \leq j$. This follows from the monotonicity of $\bar{F}$. Further, if η is an invariant up to j then it is inductive up to level $j-1$, and in addition $F_0 \Rightarrow \eta$ (initialization). However, the opposite implication does not necessarily hold (since the F_i sets are over-approximate).

Definition 6.1.3. Let $\bar{F}$ be an MFRS. A formula η is a *potential invariant* up to j, if the sequence $\bar{F}^u$ obtained by setting $F_i^u = F_i \wedge \eta$ for every $i \leq j$ and $F_i^u = F_i$ otherwise remains an MFRS. Note that η is (trivially) an invariant up to j w.r.t. $\bar{F}^u$.

Thus, a potential invariant η can safely be used to strengthen the F_i components, turning η into an invariant. Using the notion of a potential invariant we can now say that η is a potential invariant up to j iff it is inductive up to level $j-1$, and in addition $F_0 \Rightarrow \eta$ (initialization).

The algorithms discussed in this chapter use the following simple observation:

Lemma 6.1.4. *Let $\bar{F}$ be an MFRS and η a formula such that η is an invariant up to $j-1$ and $F_{j-1} \wedge \mathrm{TR} \Rightarrow \eta'$. Then η is a potential invariant up to j.*

Note that given that η is an invariant up to $j-1$, the requirement $F_{j-1} \wedge TR \Rightarrow \eta'$ is equivalent to requiring that η is inductive up to $j-1$ since in this case $F_{j-1} \wedge \eta \equiv F_{j-1}$.

6.1.1 SAT-based Reachability via IC3

IC3 [8] is a SAT-based model checking algorithm that, given a model M and a property AGp, computes increasingly long sequences $\bar{F}(M, p)$. The algorithm works iteratively, where at iteration k, the MFRS of length $k+1$ is extended to an MFRS of length $k+2$ by initializing the set F_{k+1} and possibly updating previous sets (with index $i \leq k+1$). The computation continues until either a counterexample is found or a fixpoint is reached (i.e. $F_{i+1} \Rightarrow F_i$ for some i).

One of the main features of IC3 is the fact that no unrolling of the transition relation is needed. We give a brief overview of how it operates. More details are given along the chapter as needed. For the exact details we refer the reader to [8].

IC3 starts by checking if $INIT \wedge \neg p$ or $INIT \wedge TR \wedge \neg p'$ is satisfiable, in which case a counterexample of length zero or one is found and the algorithm terminates. If both are unsatisfiable, F_0 is initialized to $INIT$ and F_1 is initialized to p. $\langle F_0, F_1 \rangle$ is an MFRS (it satisfies the conditions in Definition 2.4.1).

IC3 extends and updates $\bar{F}$, while strengthening the F_i's. The kth iteration starts from an MFRS $\langle F_0, \ldots, F_k \rangle$. Then F_{k+1} is initialized to p. Clearly, $F_k \Rightarrow F_{k+1}$ and $F_{k+1} \Rightarrow p$ hold. Therefore, the purpose of strengthening is to ensure that $F_k \wedge TR \Rightarrow F'_{k+1}$. This is done by checking that $F_k \wedge TR \wedge \neg p'$ is unsatisfiable. If this formula is satisfiable then a state $s \in F_k$ is retrieved from the satisfying assignment. s is a bad state since it reaches $\neg p$ (and by that violates $F_k \wedge TR \Rightarrow F'_{k+1}$). At this point, either s is reachable from $INIT$, in which case a counterexample exists, or s is unreachable and needs to be removed from F_k. In order to determine if s is reachable, IC3 checks the formula: $F_{k-1} \wedge TR \wedge s'$. If this formula is unsatisfiable, then s can be removed from F_k (since the property $F_{k-1} \wedge TR \Rightarrow F'_k$ of an MFRS holds without it as well), and the same process is repeated for other states in F_k that can reach $\neg p$ (if any). However, if $F_{k-1} \wedge TR \wedge s'$ is satisfiable, a pre-

decessor $t \in F_{k-1}$ of s is extracted and handled similarly to s in order to determine if t (which is also a bad state) is reachable from *INIT* or not. IC3 therefore moves back and forth along the F_i's, while retrieving bad states b and checking their reachability from *INIT* via local reachability checks of the form $F_i \wedge TR \wedge b'$. During this process, the F_i's are strengthened by removing bad states that are not reachable[1]. If a state in $F_0 = INIT$ is reached during the backwards traversal, then a counterexample is obtained.

Definition 6.1.5. Satisfiability checks of the form $F_i \wedge TR \wedge \eta$ (where $\mathrm{Vars}(\eta) \subseteq V \cup V'$) are called *i-reachability checks*.

6.1.2 Abstraction

As mentioned before, in this chapter we consider the "visible variables" abstraction [40], which is particularly suitable for hardware. Let $M_c = (V, U, INIT, TR)$ be a model and let $U_i \subseteq U$ be a set of state-variables. We refer to U_i as the set of "visible variables".

Given U_i, we consider an abstract model $M_i = (V_i, U_i, TR_i)$ of M_c where $TR_i = \bigwedge_{v \in U_i} (v' = f_v(V, V'))$ is an abstract transition relation, and $V_i = \{v \in V \mid v \in \mathrm{Vars}(TR_i) \vee v' \in \mathrm{Vars}(TR_i)\} \subseteq V$. Note that the behavior of invisible state variables (in $U \setminus U_i$) is nondeterministic.

We do not introduce an abstraction of *INIT* as part of M_i since we always consider the concrete set of initial states. M_i is an *abstraction* of M_c, denoted $M_c \preceq M_i$, in the sense that both its set of states and its transition relation are abstractions of the concrete ones, as explained next. M_i induces a set of abstract states S_i which includes all valuations to V_i. Specifically, each concrete state $s \in S$ is abstracted by the abstract state $s_i \in S_i$ that agrees with s on the assignment to the joint variables in V_i. In this case we write $s \preceq s_i$. We sometimes refer to s_i as the set of concrete states it abstracts:

[1] In fact, in order to remove a bad state b from F_i, IC3 finds a clause c that is an invariant up to i and implies $\neg b$, and adds c to F_i as a conjunct.

$\{s \in S | s \preceq s_i\}$.

In addition, TR is abstracted by TR_i in the sense that $TR \Rightarrow TR_i$. Formally, the relation $\{(s, s_i) \mid s \preceq s_i\}$ is a simulation relation from M_c to M_i.

Given an MFRS $\bar{F}(M_c, p) = \langle F_0, \ldots, F_k \rangle$ and an abstract model M_i, we say that a formula η is *inductive* up to level j w.r.t. M_i, if $F_j \wedge \eta \wedge TR_i \Rightarrow \eta'$.

Lemma 6.1.6. *Any formula inductive up to j w.r.t. M_i is also inductive up to j w.r.t. M_c.*

The lemma holds since $TR \Rightarrow TR_i$. When we do not explicitly mention a model, we refer to inductiveness w.r.t. M_c. The notion of an invariant always refers to M_c.

6.1.3 Lazy Abstraction

As mentioned above, *lazy abstraction* [36] allows to use different details of the model at different iterations of the state-space traversal. We adapt the notion of lazy abstraction to abstraction based on *visible variables* [40], and allow different variables to be visible at different time frames.

Definition 6.1.7. An *abstraction sequence* w.r.t. a model M_c is a sequence $\bar{U} = \langle U_0, \ldots, U_k \rangle$ where $U_i \subseteq U$ for $0 \leq i \leq k$, is a set of visible state-variables. $\bar{U}$ is *monotonic* if $U_i \subseteq U_{i+1}$ for each $0 \leq i < k$.

An abstraction sequence $\bar{U}$ represents different levels of abstraction of M_c. It induces a sequence of abstract models $\langle M_0, \ldots, M_k \rangle$ where M_i is defined as in Section 6.1.2. If $\bar{U}$ is monotonic, the induced sequence of abstract models is also monotonic in the sense that $M_0 \succeq \ldots \succeq M_k \succeq M_c$.

Definition 6.1.8. Let $\bar{U} = \langle U_0, \ldots, U_k \rangle$ be a monotonic abstraction sequence and $\bar{F}(M_c, p) = \langle F_0, \ldots, F_k \rangle$ an MFRS. A sequence $s_i, \ldots, s_j$ of abstract states where $0 \leq i < j \leq k + 1$ is an *abstract path from i to j* if

1. for each $i \leq l < j-1$, $(s_l, s_{l+1}) \models TR_l$, and

2. for each $i \leq l \leq \min\{j,k\}$, $s_l \cap F_l \neq \emptyset$[2].

An abstract path $s_0, \ldots, s_j$ from 0 to j is an *abstract counterexample of length* j if $s_j \cap \neg p \neq \emptyset$.

Note that the definition above is not standard. It refers to different transition relations at different steps. Also, it requires the abstract states to be part of the corresponding F_i in the sense that $s_i \cap F_i \neq \emptyset$. Unlike with concrete states, it is possible that $s_i \cap F_i \neq \emptyset$ but $s_i \not\subseteq F_i$. As a result we do not write $s_i \in F_i$.

Definition 6.1.9. An abstraction sequence $\langle {U_0}^r, \ldots, {U_k}^r \rangle$ is a *refinement* of an abstraction sequence $\langle U_0, \ldots, U_k \rangle$ if $U_i \subseteq {U_i}^r$ for each i.

6.2 Lazy Abstraction and IC3

In this section we describe our proposed algorithm for lazy abstraction, called L-IC3. The key ingredients of L-IC3 are an *abstraction sequence* $\bar{U}$ that induces different abstractions at different time frames as well as an *MFRS* $\bar{F}$.

L-IC3 starts with an initialization step and then works in *stages* (Figure 6.1). Its initialization (lines 2-5) is similar to the initialization of IC3 with one exception. If no counterexample of length 0 or 1 exists, then in addition to initializing $\bar{F}$ to $\langle F_0 = INIT, F_1 = p \rangle$, it initializes $\bar{U}$ to $\langle U_0 = \text{Vars}(p) \rangle$. Clearly, after initialization, $\bar{F}$ is an MFRS.

Each L-IC3 stage (lines 6-10) consists of an abstract model checking step and a refinement step, both performed by variations of IC3. $\bar{U}$ and $\bar{F}$ are updated in both steps.

[2]Note that s_l is an abstract state thus representing a set of concrete states. Therefore, an intersection ($\cap$) is used. Also, requirement (2) dismisses paths that are known to be spurious based on $\bar{F}$. $\min\{j,k\}$ is used for the case where $j = k+1$, in which nonempty intersection is required only up to k.

```
1: function L-IC3(p)
2:     F̄ = ⟨INIT, p⟩; Ū = ⟨Vars(p)⟩
3:     if INIT-IC3(F̄, Ū, p) == cex then
4:         return cex
5:     end if
6:     while A-IC3(F̄, Ū) == abs-cex do
7:         if REFINE(F̄, Ū) == cex then
8:             return cex
9:         end if
10:    end while
11:    return fixpoint
12: end function
```

Figure 6.1: L-IC3

The abstract model checking A-IC3 gradually extends and updates the MFRS $\bar{F}$ by adding F_i sets and strengthening the components of $\bar{F}$ until either a fixpoint is reached, or an abstract counterexample is found (line 6). In the former case, the property is proved and L-IC3 terminates (line 11). In the latter case, the counterexample is *abstract* since it is computed w.r.t. the abstract transitions. However, it is also restricted by $\bar{F}$ (see Definition 6.1.8). A refinement is then performed (line 7). If the refinement finds a concrete counterexample then it terminates. Otherwise it refines $\bar{U}$ and updates $\bar{F}$ into an MFRS (of the same length). A new L-IC3 stage (line 6) of abstraction-refinement then begins, invoking A-IC3 with the updated $\bar{F}$ and the refined $\bar{U}$.

Both the abstract model checking and the refinement update the components of the MFRS, however only the abstract model checking extends it by adding sets. The abstract model checking step also adds new components to the abstraction sequence when needed, but they are simply initialized to the last component in the sequence. Existing components of the abstraction sequence are updated (refined) only by the refinement.

Altogether, an invocation of L-IC3 results in either a fixpoint (in which

case the property is proved) or a concrete counterexample.

Iterations of L-IC3 The stages of L-IC3 should not be confused with the iterations of IC3 as a stage may extend $\bar{F}$ by more than one set. Similarly to IC3, we define an *iteration of L-IC3* to include the effort involved in the extension of $\bar{F}$ by *one* set. In iteration k, $\bar{F}$ is extended from $\langle F_0, \ldots, F_k \rangle$ to $\langle F_0, \ldots, F_k, F_{k+1} \rangle$. If no abstract counterexample is found, the iteration is performed in full by A-IC3. In fact, several iterations can be performed by a single invocation of A-IC3 (within a single stage of L-IC3), extending $\bar{F}$ by several sets (as long as no abstract counterexample is found). When an abstract counterexample is found, the corresponding iteration that starts at A-IC3 continues at the refinement step.

6.2.1 Abstract Model Checking via A-IC3

The abstract model checking algorithm, A-IC3 (Figure 6.2), either finds an abstract counterexample (line 22), or reaches a fixpoint (line 26) by computing an MFRS $\bar{F}$.

Using different abstractions The computation of $\bar{F}$ is done using a variation of IC3 which considers a *sequence of abstract* models, induced by a monotonic abstraction sequence $\bar{U} = \langle U_0 \ldots, U_k \rangle$. A-IC3 uses the abstraction sequence $\bar{U}$ and extends it as necessary, but does not change its existing components. Both abstract transition relations and abstract states are used.

Recall that IC3 performs i-reachability checks of the form $F_i \wedge TR \wedge \eta$. A-IC3 also performs these checks (within STRENGTHEN, line 20), but instead of using the concrete TR it uses the *abstract* TR_i. This means that when traversing the model's state space, A-IC3 uses different abstract transition relations at different time frames. Further, when $F_i \wedge TR_i \wedge \eta$ is satisfiable, A-IC3 retrieves an *abstract* state $s_a \in M_i$ from the satisfying assignment. This abstract state is either used to strengthen $\bar{F}$, or it is part of an abstract counterexample.

```
13: function A-IC3(F̄, Ū)
14:     k = |F̄| − 1
15:     while F̄.fixpoint() == false do
16:         U_k = U_{k−1}
17:         Ū.add(U_k)
18:         F_{k+1} = p
19:         F̄.add(F_{k+1})
20:         result = STRENGTHEN(F̄, Ū, k)
21:         if result == abs-cex then
22:             return abs-cex
23:         end if
24:         k = k + 1
25:     end while
26:     return fixpoint
27: end function
```

Figure 6.2: A-IC3

On the other hand, when A-IC3 strengthens some set F_{i+1} by adding to it a clause c which is initial (i.e. $F_0 \Rightarrow c$) and inductive at M_i up to i, i.e. $F_i \wedge c \wedge TR_i \Rightarrow c'$, then the clause c is added to all F_j such that $j \leq i$, even though it might not be inductive w.r.t. to TR_j. The justification is that c is inductive w.r.t. the concrete model (as $TR \Rightarrow TR_i$). As a result, even though abstract models are used, the obtained MFRS satisfies the requirements of Definition 2.4.1, which refer to the concrete transition relation TR. It does not necessarily satisfy the requirements of Definition 2.4.1 w.r.t. the abstract transition relations. To emphasize this, we sometimes refer to the sequence as a *concrete* MFRS.

Incrementality A-IC3 is an iterative algorithm. The iterations of L-IC3 and A-IC3 coincide, except that when an abstract counterexample is found, an iteration of L-IC3 consists of an iteration of A-IC3 followed by refinement. If A-IC3 finds a counterexample at iteration k it returns. After refinement (line 7) A-IC3 is re-invoked with an updated $\bar{F}$ that is an MFRS of the same

length. The computation of $\bar{F}$ resumes from iteration $k+1$ (line 14)[3].

Iterations In iteration $k \geq 1$, the MFRS $\langle F_0, \ldots, F_k \rangle$ and the abstraction sequence $\langle U_0, \ldots, U_{k-1} \rangle$ are extended by 1 and updated as follows (see Figure 6.2).

1. Check if a fixpoint is reached. If not:
2. U_k is initialized to U_{k-1} and added to $\bar{U}$.
3. F_{k+1} is initialized to p and added to $\bar{F}$.
4. The sets $F_0, \ldots, F_{k+1}$ are strengthened iteratively until $\langle F_0, \ldots, F_{k+1} \rangle$ becomes an MFRS, or an abstract counterexample is found.

Note that if no counterexample is found, then an iteration of A-IC3 and an iteration of L-IC3 coincide. However, if an abstract counterexample is found, then the corresponding iteration of L-IC3 includes the iteration of A-IC3 as well as the following refinement step.

Below we describe items 2 and 4 in more detail.

(2) Extending $\bar{\mathbf{U}}$: U_k is initialized to U_{k-1} (line 16). This is aimed at immediately eliminating from TR_k spurious transitions that lead from states in $F_{k-1} \subseteq F_k$ to $\neg p$ and were already removed from TR_{k-1}. Note that this initialization does not imply that the U_i sets will always be equal, since refinement might change them in different ways.

(4) Iterative Strengthening of $\bar{\mathbf{F}}$: At the beginning of the iteration, $\langle F_0, \ldots, F_k \rangle$ is a *concrete* MFRS. However, the addition of $F_{k+1} = p$ might cause the implication $F_k \wedge TR \Rightarrow F_{k+1}$ not to hold. When considering the abstract models and transition relations (as does A-IC3) this means that $F_k \wedge TR_k \Rightarrow F_{k+1}$ does not hold, i.e., there exists a bad abstract state at F_k that reaches $\neg F_{k+1} = \neg p$. To ensure that $F_k \wedge TR \Rightarrow F_{k+1}$, A-IC3 attempts to eliminate this state from F_k (even if in fact it only violates the abstract

[3]An abstract counterexample is found w.r.t. $\bar{F} = \langle F_0, \ldots, F_{k+1} \rangle$ produced in iteration k, where $|\bar{F}| = k+2$. When A-IC3 is re-invoked, k is set to $|\bar{F}| - 1 = k+1$.

implication and not the concrete one). To do so, A-IC3 first makes sure that there is no abstract predecessor of the bad state in F_{k-1}. If there is one, then A-IC3 also tries to eliminate it since it contradicts the implication $F_{k-1} \wedge TR_{k-1} \Rightarrow F_k$ (and hence potentially violates $F_{k-1} \wedge TR \Rightarrow F_k$), and so on. Each of these states is also a bad abstract state that reaches $\neg p$ along an abstract path in $F_0, \ldots, F_k$. In this sense, A-IC3 obtains an MFRS of length $k+1$ by strengthening the F_i's to exclude bad abstract states that reach $\neg p$ along an abstract path in $F_0, \ldots, F_k$. A sequence of such bad states of length $k+1$ is an abstract counterexample of length $k+1$.

A-IC3 can also be viewed as trying to eliminate all (suffixes) of abstract counterexamples of length $k+1$ w.r.t. $\langle F_0, \ldots, F_k \rangle$. From this point of view, A-IC3 identifies abstract states that might be a part of an abstract counterexample at a certain time frame, and attempts to block them by learning corresponding invariants. Recall that the abstract counterexamples we consider are restricted not only by the abstract transition relations, but also by the F_i sets (Definition 6.1.7).

Technically, bad abstract states are described by abstract proof obligations (similarly to the notion of proof obligations used in IC3).

Definition 6.2.1. An *abstract proof obligation*, or an *obligation* in short, is a pair (s_a, n) consisting of a level $n \leq k$ and an abstract state s_a such that

1. s_a is a "bad state" that reaches $\neg p$ along some abstract path ,
2. $\neg s_a$ is an invariant up until n,
3. $s_a \cap F_{n+1} \neq \emptyset$, and
4. F_n reaches s_a in one step of TR_n.

Thus $n+1$ is the minimal level intersecting s_a, and F_n reaches s_a in one *abstract* step. Note that it is possible that F_n cannot reach s_a along the concrete transitions. A-IC3 maintains two sets of obligations - *may* and *must*.

Definition 6.2.2. An obligation (s_a, n) is a *must obligation* w.r.t. iteration k if s_a must be shown unreachable from F_n in one step w.r.t. TR_n, in order to ensure that no abstract counterexample of length $k+1$ exists. All other obligations are *may obligations* w.r.t. k.

If s_a can reach $\neg p$ via an abstract path from level $n+1$ to level $k+1$, then (s_a, n) is a must obligation: unless s_a is blocked from F_{n+1} (by removing from F_n all states that reach s_a in one step), an abstract counterexample of length $k+1$ would exist. The same violation may also be reached from s_a in later levels F_j, $n+1 < j \leq k+1$, in which case it will be a suffix of a longer abstract counterexample with a longer prefix up to s_a. Therefore, we may also want to block s_a in F_j, $n+1 < j \leq k+1$. However, since different abstract transition relations are considered at each level, it is also possible that the same path leading from s_a to $\neg p$ is not valid from level $j > n+1$ since, for example, $U_j \supset U_{n+1}$ and hence the first transition along the path does not satisfy TR_j. In this case, a longer counterexample is not a valid abstract path since its suffix is not valid. The attempt to block a state s_a that is known to reach a violation from level $n+1$ in levels greater than $n+1$ creates *may* obligations[4].

The may obligations are not *required* to be blocked, but blocking them can prevent A-IC3 from encountering the same obligations/states in future iterations. On the other hand, if we report an abstract counterexample based on a may obligation, it is possible that no real abstract counterexample exists, resulting in an unnecessary refinement step which can damage the efficiency of the algorithm. We therefore greedily try to handle may obligations and strengthen $\bar{F}$ accordingly, but refrain from reporting abstract counterexamples based on them. Note that in the latter case, if the may obligation is in fact a must w.r.t. some greater k, then it will reappear as a must obligation in the following iterations.

[4]IC3 does not make a distinction between may and must obligations and handles them all the same since in the concrete case, a longer counterexample is always a *valid* path (its suffix reaching a violation is always valid).

In order to handle an obligation (s_a, n) and show s_a to be unreachable from F_n in one step, A-IC3 attempts to strengthen F_n by extracting predecessors t_a of s_a that satisfy $F_n \wedge TR_n \wedge s'_a$, defining new proof obligations based on them, and handling these obligations (by the same procedure). If F_n is successfully strengthened s.t. $F_n \wedge TR_n \wedge s'_a$ becomes unsatisfiable, then $\neg s_a$ becomes an invariant up to $n+1$. s_a is blocked by strengthening $F_0, \ldots, F_{n+1}$ accordingly.

Key procedures used by A-IC3 are described in Section 6.2.2.

6.2.2 Detailed Description of Strengthening

We now describe the procedures used by A-IC3 in detail.

Strengthen (Figure 6.3)

STRENGTHEN starts by checking $F_k \wedge TR_k \wedge \neg p'$ (line 29). If it is unsatisfiable, then $F_k \wedge TR \wedge \neg p'$ is unsatisfiable as well (since $TR \Rightarrow TR_k$). Thus $\bar{F}$ is already an MFRS and no further strengthening is needed.

Assume $F_k \wedge TR_k \wedge \neg p'$ is satisfiable. An abstract state $s_a \in M_k$ that reaches $\neg p$ in one abstract step is extracted from the satisfying assignment, meaning $s_a \cap F_k \neq \emptyset$. All concrete states in $s_a \cap F_k$ can reach $\neg p$ via TR_k and therefore, if the property is to be proven, s_a must be blocked in F_k. Otherwise, an abstract counterexample exists.

In order to block s_a in F_k, STRENGTHEN calls BLOCKSTATE on the bad state s_a at level k (line 32).

Lemma 6.2.3. *s_a satisfies the requirements of* BLOCKSTATE *at level k.*

Proof. Clearly, s_a is a "bad state" at level k as it reaches $\neg p$ in one abstract step and $s_a \cap F_k \neq \emptyset$, since s_a was retrieved from a satisfying assignment to $F_k \wedge TR_k \wedge \neg p'$, i.e. a satisfying assignment to F_k. Furthermore, $\neg s_a$ is an invariant up to $k-1$, otherwise the same violation would have been found and eliminated in the previous iteration. □ □

```
28: function STRENGTHEN(F̄,Ū,k)
29:     while F_k ∧ TR_k ∧ ¬p' == SAT  do
30:         obligations = ∅
31:         retrieve abstract predecessor s_k
32:         if BLOCKSTATE(F̄,s_k,k,k,must) == abs-cex then
33:             return abs-cex
34:         end if
35:         while obligations ≠ ∅ do
36:             ((s_a, n), handleMay) = CHOOSENEXT(obligations)
37:             if F_n ∧ TR_n ∧ s'_a == SAT then
38:                 retrieve abstract predecessor t_n
39:                 if BLOCKSTATE(F̄,t_n,n,k,must) == abs-cex then
40:                     if handleMay then
41:                         obligations.clearAllMust()
42:                     else
43:                         return abs-cex
44:                     end if
45:                 end if
46:             else
47:                 obligations.removeMust(s_a,n)
48:                 BLOCKSTATE(F̄,s_a,n + 2,k,may)
49:             end if
50:         end while
51:     end while
52:     PROPAGATECLAUSES(F̄)
53:     return done
54: end function
```

Figure 6.3: Iterative strengthening of A-IC3

BlockState either finds a counterexample or initializes the set(s) of obligations to reflect the need to block s_a (and possibly adds invariants to the F_i's).

Strengthen then handles the proof obligations one at a time. ChooseNext (line 36) first considers obligations from the must set only. Obligations are chosen in increasing order of their time frames. If the must set becomes empty, then as long as the may set is not empty, one may obligation with a minimal time frame is moved from the may set to the must set. Strengthen then continues, with the exception that counterexamples are no longer reported.

Given a proof obligation (s_a, n):

- If F_n can indeed reach s_a in one (abstract) step, i.e., $F_n \wedge TR_n \wedge s'_a$ is satisfiable, then a predecessor t_a of s_a s.t. $t_a \cap F_n \neq \emptyset$ is extracted from the satisfying assignment (line 38). By Lemma 6.2.4 (see below), $t_a \cap F_i = \emptyset$ for all $i < n$. Thus $\neg t_a$ is an invariant up to $n-1$. Next, the state t_a needs to be blocked (eliminated) from level $l = n$ (line 39).

- When $F_n \wedge TR_n \wedge s'_a$ becomes unsatisfiable, the proof obligation (s_a, n) is removed (line 47) since s_a can no longer be reached from level n. In fact, $\neg s_a$ is now a potential invariant up to level $n+1$ (see Lemma 6.2.6 below).In order not to encounter s_a in later iterations, we speculatively attempt to block (eliminate) s_a from level $l = n+2$, while using the *may* parameter (line 48). The call to BlockState also adds $\neg s_a$ (or a stronger clause) as an invariant (line 71).

Lemma 6.2.4. *Let (s_a, n) be a proof obligation, and let t_a be an abstract state such that $(t_a, s_a) \models \mathrm{TR}_n$. Then $t_a \cap F_i = \emptyset$ for every $i \leq n-1$.*

Proof. Let (s_a, n) be a proof obligation. In particular, $s_a \cap F_{n+1} \neq \emptyset$. Suppose further that $(t_a, s_a) \models TR_n$. We show that $t_a \cap F_i = \emptyset$ for every $i \leq n-1$. Since $F_i \subseteq F_{n-1}$ for every $i \leq n-1$, it suffices to show that $t_a \cap F_{n-1} = \emptyset$.

At the beginning of the $n-1$'th L-IC3-iteration (which added F_n), it was the case that $s_a \cap F_n \neq \emptyset$. This is because F_n was initialized to p, and clearly $s_a \cap p \neq \emptyset$ (since $F_{n+1} \subseteq p$ and $s_a \cap F_{n+1} \neq \emptyset$). On the other hand, at the current L-IC3-iteration $s_a \cap F_n = \emptyset$, since for a proof obligation (s_a, n), $\neg s_a$ is an invariant up to n.

This means that there is a set of clauses C that were added to F_n such that $p \cap \bigcap C \cap s_a = \emptyset$ and for every $c \in C$, $s_a \not\subseteq c$ or equivalently $s_a \cap \neg c \neq \emptyset$ (other clauses are not considered as they do not contribute to blocking s_a anyway). Every clause c added to F_n is inductive at some time frame $\geq n-1$ (see Lemma **??**). Therefore, for every $c \in C$ there is a frame $i_c \geq n-1$ such that $F_{i_c} \wedge c \wedge TR_{i_c} \Rightarrow c'$. Furthermore, since $s_a \cap F_{n+1} \neq \emptyset$, at least one of these clauses was not added to F_{n+1} which ensures that there is some $c \in C$ such that $i_c \leq n-1$ (recall that when c is inductive at i_c it is added up to $i_c + 1$), i.e. $i_c = n-1$. We denote such a clause by c_0. We have that $F_{n-1} \wedge c_0 \wedge TR_{n-1} \Rightarrow c_0'$, or equivalently $F_{n-1} \wedge c_0 \wedge TR_{n-1} \wedge \neg c_0' == UNSAT$.

Now assume to the contrary that $t_a \cap F_{n-1} \neq \emptyset$. To reach a contradiction we first note that since $(t_a, s_a) \models TR_n$, it is also the case that $(t_a, s_a) \models TR_{n-1}$. Therefore by our assumption we have that $(t_a, s_a) \models F_{n-1} \wedge TR_{n-1} \equiv F_{n-1} \wedge c_0 \wedge TR_{n-1}$. The equivalence is since c_0 is a clause in F_{n-1}. Together with the property that $s_a \cap \neg c_0 \neq \emptyset$ (since $c_0 \in C$ and by the choice of C), we have that (t_a, s_a) is a satisfying assignment for $F_{n-1} \wedge c_0 \wedge TR_{n-1} \wedge \neg c_0'$, in contradiction to the property that it is $UNSAT$. □ □

Lemma 6.2.5. *Let (s_a, n) be a proof obligation, and let t_a be an abstract state such that $(t_a, s_a) \models \mathrm{TR}_n$. Then t_a satisfies the requirements of* BlockState *from level $l = n$.*

Proof. First, t_a is a "bad state" at level n as it reaches s_a in one abstract step and s_a itself, being a part of a proof obligation, is a "bad state" that reaches $\neg p$ along an abstract path. In addition, $t_a \cap F_n \neq \emptyset$, since t_a was retrieved from a satisfying assignment to $F_n \wedge TR_n \wedge s_a'$, i.e. a satisfying assignment to F_n. Last, by Lemma 6.2.4 $\neg t_a$ is an invariant up to $n-1$. □ □

```
55: function BlockState(F̄,t_a,l,k,type)
56:     if l > k + 1 then
57:         min = k + 1
58:     else
59:         min = FindNonInductive(F̄,¬t_a,l − 1,k)
60:         if min == 0 then
61:             return abs-cex
62:         end if
63:         if min ≤ k then
64:             if type == must && min == l-1 then
65:                 obligations.addMust(t_a, min)
66:             else
67:                 obligations.addMay(t_a, min)
68:             end if
69:         end if
70:     end if
71:     AddInvariant(F̄,¬t_a,min)
72:     return done
73: end function
```

Figure 6.4: BlockState procedure of A-IC3

Lemma 6.2.6. *Let (s_a, n) be a proof obligation. If $F_n \wedge \mathrm{TR}_n \wedge s'_a$ becomes unsatisfiable then s_a satisfies the requirements of* BlockState *from level $l = n + 2$.*

Proof. First, s_a is a "bad state" at level $n+2$ as it reaches $\neg p$ in one abstract step of TR_n (but not necessarily of TR_{n+2}, which is why the *may* flag is used) and $s_a \cap F_{n+2} \neq \emptyset$, since $s_a \cap F_{n+1} \neq \emptyset$ (recall that (s_a, n) is a proof obligation) and $F_{n+1} \Rightarrow F_{n+2}$. Furthermore, by Lemma 6.1.4 since $\neg s_a$ is an invariant up to n (as a proof obligation) and $F_n \wedge TR_n \wedge s'_a$ becomes unsatisfiable, then $\neg s_a$ is a potential invariant up to $n + 1$. □ □

As explained above, a counterexample found by BlockState is reported by Strengthen iff may obligations are not yet handled (lines 33 and 43).

Remark 1. *Note that ignoring a counterexample reported by* BlockState *when it failed to block a may obligation (s_a, n) does not compromise the correctness of the algorithm, since an MFRS up to level $k + 1$ is still obtained.*

Moreover, if s_a does reach a violation from level $n+1$, which means that the same obligation is in fact required *for the property to hold, then it will reappear as a must obligation in the following iterations. In fact, even if the abstract counterexample is a real abstract counterexample, it might be worth while to defer handling it. This is because it is possible that in later iterations, where the abstraction becomes more precise, it will cease to exist, whereas a pre-mature invocation of refinement, which traverses the concrete state apace restricted by the F_i's, might be costly.*

BlockState (Figure 6.4)

BlockState($\bar{F}$,t_a,l,k,*type*) is used for blocking a "bad state" t_a from level l (i.e. $t_a \cap F_l \neq \emptyset$) up to $k+1$, where $\neg t_a$ is already known to be a potential invariant up to $l-1$. t_a is a "bad state" as at reaches $\neg p$ along some abstract path, however this path might not be from level l, in which case the *may* flag is used.

Note that if $l > k+1$ (line 57) then t_a is already blocked up to $k+1$. Thus $\neg t_a$ is added as an invariant up to $k+1$ (line 71). Otherwise, BlockState looks for a level such that $\neg t_a$ is a potential invariant up to it.

Specifically, BlockState looks for the minimal level min between $l-1$ and k s.t. $F_{min} \wedge TR_{min} \wedge t'_a$ is satisfiable (line 59) (meaning that t_a can be reached in one step from min). The important property is that $\neg t_a$ is a potential invariant up to min: If $min = l-1$, this holds since $\neg t_a$ is already known to be a potential invariant up to level $l-1$ (this is also why the search for min starts at $l-1$). If $min > l-1$, then the fact that $F_{min-1} \wedge TR_{min-1} \wedge t'_a$ is unsatisfiable implies that $\neg t_a$ is inductive at $min-1$ w.r.t. M_{min-1}, and hence, by Lemma 6.1.6 also w.r.t. M_c. Thus, it is a potential invariant up to min.

If $min = 0$, then the "bad state" t_a is reachable from $INIT$ in one step of TR_0. Thus, an abstract counterexample is reported (line 61). If $min = k+1$ then no corresponding level was found up to k, i.e., $\neg t_a$ is a potential invariant

up to $k+1$ and no new proof obligation is added. However, if $min \leq k$ is found then the pair (t_a, min) is added as a new proof obligation (lines 64-68). Either way, $\neg t_a$ is added as an invariant up to min by calling AddInvariant (line 71). AddInvariant learns an invariant that strengthens $\neg t_a$ and adds it to $F_0, \ldots, F_{min}$.

Lemma 6.2.7. *If $min \leq k$ is found then the pair (t_a, min) is a proof obligation.*

Proof. By Definition 6.2.1, we need to show that:

1. t_a is a "bad state" that reaches $\neg p$ along some abstract path,
2. $\neg t_a$ is an invariant up until min,
3. $t_a \cap F_{min+1} \neq \emptyset$, and
4. F_{min} reaches t_a in one step of TR_{min}

Item 1 is a property of the input t_a of BlockState. Items 2 and 4 hold by the choice of min, as explained above, with the addition that AddInvariant is called turning $\neg t_a$ from a potential invariant to an actual invariant up to min. Finally, $t_a \cap F_l \neq \emptyset$ (as a property of the inputs of BlockState). In addition, $min \geq l-1$ (by the choice of min), therefore $F_{min+1} \supseteq F_l$ and hence item 3 holds. □ □

Classifying obligations as may/must is performed in lines 64- 68 of BlockState. Note that only obligations of the form $(t_a, l-1)$ are must obligations. The initial obligations generated by the call BlockState($\bar{F}$,s_a,k,k,$must$) in line 32 of Strengthen whose level is exactly $k-1$ become must obligations. Later on, only obligations of the form $(t_a, n-1)$ generated by the call to BlockState($\bar{F}$,t_a,n,k,$must$) in line 39 of Strengthen when handling a must obligation (s_a, n), where t_a is a predecessor of s_a, are considered must obligations. The rest are may obligations.

AddInvariant

If for some state t_a and some level $min \leq k+1$, the formula $\neg t_a$ is a potential invariant up to level min, then AddInvariant (called from BlockState line 71) is used to add a strengthening of $\neg t_a$ to all F_j's s.t. $j \leq min$. More precisely, $\neg t_a$ is strengthened to some subclause[5] c s.t. $F_0 \Rightarrow c$ and $F_{min-1} \wedge c \wedge TR_{min-1} \Rightarrow c'$, i.e. c is inductive w.r.t. M_{min-1} and hence, by Lemma 6.1.6, also w.r.t. M_c. Consequently, c is also a potential invariant up to min, but it is a stronger invariant than $\neg t_a$ (since $c \Rightarrow \neg t_a$). The clause c is added as a conjunct to $F_0, \ldots, F_{min}$ while maintaining the properties of a (concrete) MFRS[6]. AddInvariant always finds a clause to add, since $\neg t_a$ itself satisfies the requirements.

Lemma 6.2.8. *Let $\langle F_0, \ldots, F_k \rangle$ be an MFRS. Let $n \leq k$ and let c be a clause that is inductive up to $n+1$ w.r.t. M_c. If $F_0 \Rightarrow c$ and if $F_i^* = F_i \wedge c$ for $i \leq n+1$ and $F_i^* = F_i$ otherwise, then c is an invariant up to $n+1$ and $\langle F_0^*, \ldots, F_k^* \rangle$ is also an MFRS.*

Proof. c is inductive up to $n+1$ w.r.t. M_c. Therefore, it follows that $F_i \wedge c \wedge TR \Rightarrow c'$ for $0 \leq i \leq n$. Let us define $\langle F_0^*, \ldots, F_k^* \rangle$ s.t. $F_i' = F_i \wedge c$ for $i \leq n+1$ and $F_i^* = F_i$ otherwise. Since $F_0 \Rightarrow c$, $F_0^* \equiv F_0$. Using the fact that $F_i \wedge c \wedge TR \Rightarrow c'$ for $0 \leq i \leq n$, we get that $F_i^* \wedge TR \Rightarrow c'$ for $0 \leq i \leq n$. From this it follows that $\langle F_0^*, \ldots, F_k^* \rangle$ is a MFRS. By the definition of F_i^*, it follows directly that $F_i^* \Rightarrow c$ for $0 \leq i \leq n+1$ and thus c is an invariant up to $n+1$. □ □

[5] A state t_a is represented by a conjunction of literals, which makes its negation $\neg t_a$ a clause (i.e., a disjunction of literals). A subclause of $\neg t_a$ consists of a subset of its literals.

[6] Note that while c is inductive w.r.t. M_{min-1} up to $min-1$, it is not necessarily inductive w.r.t. M_i where $i < min-1$ (in case $U_i \subset U_{min-1}$). Still, it is safely added to F_{i+1} for $i < min-1$ since it is an invariant w.r.t. M_c.

```
74: function REFINE(F̄,Ū)
75:     result = C-STRENGTHEN(F̄)
76:     if result == cex then
77:         return cex
78:     end if
79:     REFINEABSTRACTION(F̄,Ū)
80:     return done
81: end function
```

Figure 6.5: REFINE procedure of A-IC3

PropagateClauses

Similarly to IC3, if the main loop in STRENGTHEN terminates, added clauses are propagated forward by PROPAGATECLAUSES (line 52). Specifically, if $F_i \wedge c \wedge TR_i \wedge \neg c'$ is unsatisfiable then the clause c from F_i can safely be added to F_{i+1} while maintaining the properties of an MFRS. This is done in order to get to a fixpoint.

6.2.3 Refinement

If A-IC3 finds an abstract counterexample of length $k+1$, refinement is invoked by L-IC3 (line 7). Refinement either finds a concrete counterexample or eliminates *all* concrete spurious counterexamples of length $k+1$. In the latter case, refinement also refines $\bar{U}$ to ensure that no *abstract* counterexample of length $k+1$ existsBoth an updated MFRS $\bar{F}^r = \langle F_0^r, \ldots, F_{k+1}^r \rangle$ and a refined monotonic abstraction sequence $\bar{U}^r = \langle U_0^r, \ldots, U_k^r \rangle$ are returned.

The REFINE procedure is described in Figure 6.5. REFINE first invokes C-STRENGTHEN, the strengthening procedure of the concrete IC3, on the sequence $\langle F_0, \ldots, F_{k+1} \rangle$ (whose prefix up to F_k is an MFRS) obtained from the abstract model checking. If a concrete counterexample is found the algorithm terminates (lines 75-78). Otherwise, no concrete counterexample of length $k+1$ exists. Moreover, the updated (strengthened) sets $F_0^r, \ldots, F_{k+1}^r$ comprise an MFRS. It remains to refine the abstraction sequence $\bar{U}$ in or-

der to eliminate all *abstract* counterexamples of length $k+1$ as well. Thus, RefineAbstraction is invoked (line 79).

RefineAbstraction

A-IC3 found an abstract counterexample since it failed to strengthen the F_i's. Meaning, the relevant i-reachability checks $F_i \wedge TR_i \wedge t'_a$ (or $F_k \wedge TR_k \wedge \neg p'$) could not be made unsatisfiable when using TR_i. C-Strengthen, on the other hand, succeeds to do so. Namely, for each i-satisfiability check $F_i \wedge TR_i \wedge t'_a$ (resp. $F_k \wedge TR_k \wedge \neg p'$) of A-IC3 that was satisfiable, C-Strengthen manages to make the corresponding check $F^r_i \wedge TR \wedge t'$ for each $t \preceq t_a$ (resp. $F^r_k \wedge TR \wedge \neg p'$) unsatisfiable, either by strengthening F^r_i or simply since it considers TR. Moreover, once $F^r_i \wedge TR \wedge t'$ becomes unsatisfiable, C-Strengthen derives from it a clause $c \Rightarrow \neg t$ s.t. c is inductive up to i, i.e. $F^r_i \wedge c \wedge TR \Rightarrow c'$ holds. C-Strengthen strengthens $\bar{F}^r$ by adding c (invariant) as a new clause in all sets up to F^r_{i+1}. We consider it a *learned clause* at level $i+1$. To handle clauses that were propagated forward, we consider them learned clauses at the highest level in which they were added (instead of at the level in which they were actually learned). Recall that propagation to level $i+1$ also takes place only after checking that the clause is inductive up to i. The purpose of RefineAbstraction is to ensure that for a learned clause c at level $i+1$, $F^r_i \wedge c \wedge TR^r_i \Rightarrow c'$ (with TR^r_i instead of TR) also holds. Meaning, c is inductive up to i w.r.t. (the refined) M^r_i.

Lemma 6.2.9. *Let c be a clause learned by* C-Strengthen *at level $i+1$. If $F^r_i \wedge \mathrm{TR}^r_i \Rightarrow {F^r_{i+1}}'$ then $F^r_i \wedge c \wedge \mathrm{TR}^r_i \Rightarrow c'$.*

Proof. Consider a learned clause c at level $i+1$. Assume that $F^r_i \wedge TR^r_i \Rightarrow {F^r_{i+1}}'$. Since c is a learned clause at level $i+1$, then by the property of IC3, c was added to both F^r_i and F^r_{i+1}. As a result, it holds that $F^r_i \wedge c \equiv F^r_i$ and $F^r_{i+1} \Rightarrow c$. Therefore, we have that $F^r_i \wedge c \wedge TR^r_i \Rightarrow c'$. □ □

Based on the previous lemma, in order to ensure $F^r_i \wedge c \wedge TR^r_i \Rightarrow c'$, it

suffices to ensure unsatisfiability of $F_i^r \wedge TR_i^r \wedge \neg {F_{i+1}^r}'$ for every level $i+1$ in which learned clauses exist.

In addition, to handle the possibility that $F_k \wedge TR_k \wedge \neg p'$ is satisfiable while $F_k \wedge TR \wedge \neg p'$ is unsatisfiable (without any strengthening), refinement also makes sure that $F_k^r \wedge TR_i^r \wedge \neg {F_{k+1}^r}'$ becomes unsatisfiable. This addresses the case where an abstract counterexample was reported, however the last abstract transition along the counterexample admits no corresponding concrete transition, hence refinement does not update the MFRS (i.e., $F_i^r = F_i$ for every $0 \leq i \leq k+1$) and no learned clauses exist.

To ensure unsatisfiability of a formula $F_i^r \wedge TR_i^r \wedge \neg {F_{i+1}^r}'$, we consider the same formula over TR, which is clearly unsatisfiable. We derive from it an unSAT-core. The next-state variables that appear in the unSAT-core, denoted $NS(\text{unSatCore}) = \{v \in V \mid v' \in \text{Vars(UnSatCore)}\}$, are added to U_i.

Lemma 6.2.10. *Let $F_i^r \wedge \text{TR} \wedge \eta'$ be an unsatisfiable formula and let UnSatCore be its unsat core. Let $U_i^r \supseteq NS(\text{UnSatCore})$. Then $F_i^r \wedge \text{TR}_i^r \wedge \eta'$ is unsatisfiable.*

Finally, we propagate variables that were added to U_i^r forward in order to obtain a monotonic abstraction sequence. Since we only add variables to U_i^r, i.e. make the transition relation TR_i^r more precise, then the corresponding formulas remain unsatisfiable.

As an immediate conclusion of Lemma 6.2.9 and Lemma 6.2.10, the refinement of $\bar{U}$ ensures the following property which is important for the correctness of the algorithm:

Lemma 6.2.11. *Let $\bar{F}^r = \langle F_0^r, \ldots, F_{k+1}^r \rangle$ be the updated MFRS, and let $\bar{U}^r = \langle U_0^r, \ldots, U_k^r \rangle$ be the refined abstraction sequence. Then, for any clause c that was added to the MFRS, if c was learned at level $i+1$ then c is inductive up to i w.r.t. (the refined) M_i^r.*

6.2.4 Correctness Arguments

The MFRS obtained by L-IC3 is concrete. Specifically, it does not necessarily satisfy $F_i \wedge TR_i \Rightarrow F_{i+1}$. This results both from refinement that adds invariants learned based on the concrete TR, and from A-IC3 that learns an invariant based on some TR_i, but also adds it to F_{j+1} for $j < i$ even if it is not inductive w.r.t. TR_j. This complicates the correctness proof.

In particular, in IC3, when a proof obligation (s, n) is handled, then for any predecessor t of s, $\neg t$ is an invariant up to $n-1$, otherwise s would belong to a lower frame (since $F_i \wedge TR \Rightarrow F_{i+1}$). Now consider an abstract proof obligation (s_a, n). If we assume to the contrary that the predecessor t_a intersects some F_i (for $i < n$) then we can still deduce that the transition $(t_a, s_a) \models TR_n$ also exists at a lower frame, i.e. $(t_a, s_a) \models TR_i$ for $i < n$. This is since $TR_n \Rightarrow TR_i$ (recall that the same does not necessarily hold for $i > n$). However, we cannot immediately deduce that $s_a \cap F_{i+1} \neq \emptyset$ since $F_i \wedge TR_i \Rightarrow F_{i+1}$ might *not* hold. It turns out that this property does hold (see Lemma 6.2.4), but more complicated arguments are needed, based on the following:

Lemma 6.2.12. *Let $\bar{F} = \langle F_0, \ldots, F_{k+1} \rangle$ and $\bar{U} = \langle U_0, \ldots, U_k \rangle$ be the sequences obtained at the end of the k'th iteration of L-IC3, i.e. either at the end of a refinement step or at the end of an iteration of A-IC3 in the case that no counterexample was found. Then*

1. *$\bar{F}$ is an MFRS.*
2. *For every clause c that was added to some F_i in $\bar{F}$ there exists some $j \geq i-1$ s.t. c is inductive up to j w.r.t. M_j.*
3. *No abstract counterexample of length $k+1$ exists w.r.t. the prefix $\langle F_0, \ldots, F_k \rangle$ of $\bar{F}$.*

Proof. The proof is inductive. Consider an iteration of L-IC3. It consists of an iteration of A-IC3 adding F_{k+1}, possibly followed by a refinement step.

We first show that for every clause c that was added to some F_i in $\bar{F}$ during the above, there exists some $j \geq i-1$ s.t. c is inductive up to j w.r.t. M_j. An important property to note is that during the run of the algorithm, the F_i sets, as well as the TR_i transition relations are only strengthened (resp., refined). Therefore, if at some point during the algorithm a clause c is inductive up to j w.r.t. M_j for some j, meaning that $F_j \wedge c \wedge TR_j \Rightarrow c'$ holds, then c will remain inductive up to j w.r.t. M_j later on as well, since $F_j \wedge c \wedge TR_j \Rightarrow c'$ will keep holding with the strengthened sets (and refined transition relations). This is because the strengthening only strengthens the left hand side and hence does not damage the implication. As a result, it suffices to show that every clause c that was added to some F_i in $\bar{F}$ is inductive up to some $j \leq i-1$ w.r.t. M_j, *at some point* during the iteration.

If a clause c is added by A-IC3 to F_i, then it is either added by calling AddInvariant at some level $min \geq i$ or by calling PropagateClauses at level $min = i$. In both cases c is inductive at level $min - 1 \geq i - 1$ w.r.t. TR_{min-1} when it is added. If a clause in F_i is added during refinement, then there is some level $j + 1 \geq i$ where it is a learned clause. Thus by Lemma 6.2.11 at the end of the refinement step it is inductive up to $j \geq i-1$ w.r.t. (the refined) M_j.

We now show that the obtained $\bar{F}$ is an MFRS. $F_0 = INIT$ holds due to the initialization. Similarly, $F_i \Rightarrow p$ holds due to the initialization of the F_i sets to p, and due to the property that the sets are only strengthened later on. $F_i \Rightarrow F_{i+1}$ holds when F_{i+1} is initialized (since it is initialized to p and $F_i \Rightarrow p$), and continues to hold since any clause that is added to F_{i+1} is also added to F_i. Finally, it remains to show that $F_i \wedge TR \Rightarrow F'_{i+1}$. We show this in two parts. First, we show that it holds at the end of the L-IC3 iteration (possibly including a refinement step) that added F_{i+1} to the MFRS. Next, we show that later updates of F_i and F_{i+1} maintain this property.

To show that $F_i \wedge TR \Rightarrow F'_{i+1}$ holds at the end of the iteration that added F_{i+1} to the MFRS, we recall that F_{i+1} is initialized to p and we note

that the termination condition of an iteration of A-IC3 is that $F_i \wedge TR_i \wedge \neg p' == UNSAT$ (see STRENGTHEN), meaning that $F_i \wedge TR_i \Rightarrow p' = F'_{i+1}$. Moreover, since $TR \Rightarrow TR_i$, the latter implies that $F_i \wedge TR \Rightarrow p' = F'_{i+1}$. Similarly, the termination condition of the refinement step (if applicable) is that $F_i \wedge TR \Rightarrow p' = F'_{i+1}$. To show that later updates of F_i and F_{i+1} maintain this property, we rely on the property that any clause added to F_{i+1} is inductive up to some $j \geq i$ w.r.t. M_j. This means, that it is also inductive w.r.t. to M_c. Therefore, $F_i \wedge c \wedge TR \Rightarrow c'$. Since c is added both to F_i and to F_{i+1}, the property $F_i \wedge TR \Rightarrow F'_{i+1}$ is maintained.

It remains to show that no abstract counterexample of length $k+1$ exists w.r.t. the prefix $\langle F_0, \ldots, F_k \rangle$ of $\bar{F}$. If refinement was not needed then this holds trivially since the termination condition of STRENGTHEN is that $F_k \wedge TR_k \wedge \neg p == UNSAT$.

□ □

In particular, this means that the clauses added to the last set of the sequence, F_{k+1}, are inductive up to k w.r.t. M_k, hence at the end of the k'th iteration of L-IC3 adding F_{k+1} it holds that $F_k \wedge TR_k \Rightarrow F_{k+1}$ (recall that the same does not necessarily hold for $i < k$).

Theorem 6.2.13. *L-IC3 either terminates with a fixpoint, in which case the property holds, or with a concrete counterexample.*

6.2.5 Monotonicity of the Abstraction Sequence

Monotonicity of the abstraction sequence ensures that when A-IC3 attempts to block a state t_a that reaches a violation at level n, then $\neg t_a$ is necessarily an invariant up until $n-1$ (see Lemma 6.2.4). Recall that if some state t_a reaches a violation from step n along the abstract transitions, it is not guaranteed that the same violation can be reached from t_a at level $i > n$. However, the fact that for each $i < n$, $U_i \subseteq U_n$, and as a result $TR_n \Leftarrow TR_i$, ensures that the same violation *can* be reached from t_a at any level $i < n$.

This ensures that $\neg t_a$ is an invariant up until $n-1$, otherwise, the violation would have been found in previous iterations.

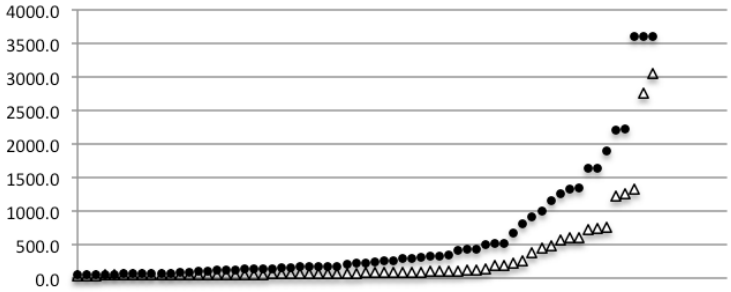
The same property does not hold if a non-monotonic abstraction sequence is used, which means that in this case deducing that $\neg t_a$ is an invariant up to $n-1$ when attempting to block a state t_a at level n is simply incorrect.

Another motivation for the monotonicity of the abstraction sequence is the following. Recall that $F_i \subseteq F_{i+1}$ for each i. This means that any state $t_a \in F_i$, and in particular states that reach a violation along some abstract path, will be encountered again in F_{i+1}. As a result, the same information needed to show that $t_a \in F_i$ cannot reach a violation from level i is likely to be needed to show that t_a cannot reach a violation from F_{i+1} as well. Restricting the discussion to monotonic abstraction sequences automatically ensures that if the abstract transition relations carry enough information to refute all violations starting at states from F_i, then the same holds when considering the same states in F_{i+1}. While it is possible that a different abstraction can be used to refute the existence of a violation from $i+1$, in most cases the effort of computing this abstraction (by invoking refinement multiple times) exceeds its potential benefit.

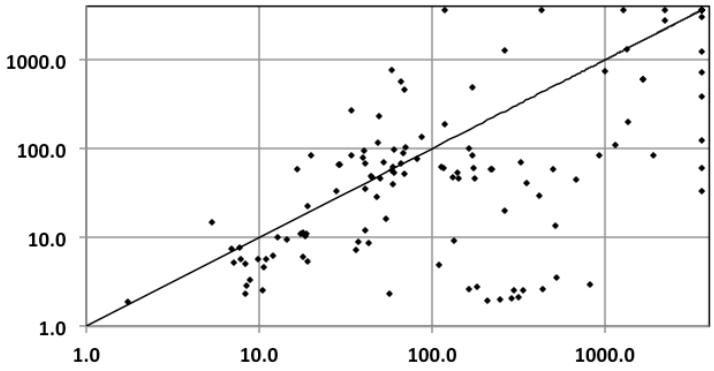
6.3 Experimental Results

For the implementation of the two algorithms we collaborated with *Jasper Design Automation*[7]. We used Jasper's formal verification platform in order to implement both the original IC3 and our L-IC3 algorithm. In both implementations we used optimizations from [28] (such as ternary simulation). Implementing these algorithms using Jasper's platform allowed us to develop and experiment with various real-life industrial designs and properties from various major semiconductor companies. All designs contain thousands of state variables in the cone of influence of the properties.

[7] An EDA company: http://www.jasper-da.com



(a) Runtime trend. Dots represent IC3, triangles represent L-IC3. Test-cases are sorted in an increasing runtime order.



(b) Comparing runtime. IC3 on X-axis and L-IC3 on Y-axis

Figure 6.6: Runtime information for L-IC3 and IC3

The timeout was set to 3600 seconds and experiments were conducted on systems with Intel Xeon X5660 running at 2.8GHz and 24GB of main memory.

We experimented with 122 real safety properties from different designs. Figure 6.6 shows two different analyses comparing the runtime of L-IC3 and IC3. Runtime trends are shown in Figure 6.6a. As can be seen, the overall trend is in favor of L-IC3. In Figure 6.6b runtime for IC3 and L-IC3 is represented by the X-axis and Y-axis respectively. We can clearly see the advantage of using L-IC3 on the more complicated test cases. These test cases are represented by the dots that are below the diagonal by a big margin. On these examples, the improvement in runtime is up to ***two*** orders

of magnitude. The cases where IC3 performs better are usually cases where L-IC3 spends most of the time in refinement. Also, for false properties (counterexample exists), the performance of L-IC3 is affected by the way we treat *may* and *must* obligations. Due to our special handling, L-IC3 may lose the ability to find a counterexample which is longer than the length of the computed Ω. In those cases, IC3 may perform better. Note that the scatter at the middle is a bunch of comparable properties where both algorithms are on par.

In the given timeout, 7 properties cannot be solved by IC3 but are solved by L-IC3; 5 properties cannot be solved by L-IC3 but are solved by IC3. There are also 5 properties that cannot be solved by either algorithm. The overall runtime for IC3 is 75558 seconds while for L-IC3 it is 55424 seconds.

The laziness of our abstraction-refinement algorithm is demonstrated in Table 6.1. The table shows how the abstraction is refined along increasing time frames. Different frames contain different variables that are needed in order to prove or disprove the given property. This demonstrates the fact that L-IC3 indeed takes advantage of the lazy abstraction framework.

Table 6.2 presents runtime characteristics for L-IC3 and IC3. In particular, it shows the number of clauses and the number of variables in Ω when either a fixpoint or a counterexample is found. In many of the examples the number of clauses produced by L-IC3 for its Ω is significantly smaller than the number of clauses produced by IC3. Recall that each of the clauses is learned via several local reachability checks. The reduced number of clauses thus indicates that L-IC3 applies a smaller number of checks and therefore issues a smaller number of calls to the SAT solver. This can explain the speedups it obtains.

An additional reason for the speedups is the fact that the local reachability checks of L-IC3 are easier than those of IC3. This is because the abstract transition relations TR_i are much smaller (in number of variables) than TR (see table 6.1). Further, the sets F_i, computed by L-IC3 are smaller than

those computed by IC3 (see Table 6.2).

Recall that in Section 6.2.1 we distinguish between *must* and *may* obligations. The results reported above are obtained while using this distinction and handling all the *may* obligations after the *must* obligations, as described there. We also tried other configurations. For example, we ran experiments that do not distinguish between *must* and *may* obligations. Our experiments show that distinguishing between the two yields a better overall performance.

In addition to the industrial experiments, we also ran experiments on the HWMCC'11 benchmark. We used the test-cases with single properties. Most of the properties in this benchmark are fairly easy and can be solved in a matter of a few seconds both by IC3 and L-IC3. There are also a few cases where IC3 performs better or even reaches a result while L-IC3 does not. In these cases L-IC3 spends most of the time in refinement. On the other hand, there are several test cases that can only be solved by L-IC3 while IC3 reaches timeout.

N	♯Vars	Laziness - Time Frames and Number of Vars																	
		♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV	♯TF	♯AV
f_1	11866	[0-0]	323	[1-1]	647	[2-2]	686	[3-3]	699	[4-4]	705	[5-5]	713	[6-6]	714	[7-7]	728	[8-8]	743
		[9-9]	752	[10-10]	755	[11-11]	761	[12-12]	767	[13-13]	777	[14-14]	783	[15-15]	789	[16-18]	811		
f_2	5693	[0-7]	12																
f_3	5693	[0-0]	8	[1-1]	56	[2-2]	64	[3-3]	74	[4-4]	82	[5-7]	91						
f_4	5693	[0-6]	31	[7-7]	42	[8-8]	51	[9-13]	54										
f_5	5773	[0-0]	260	[1-1]	381	[2-2]	401	[3-3]	419	[4-34]	430								
f_6	1183	[0-0]	185	[1-1]	248	[2-2]	255	[3-3]	259	[4-4]	262	[5-5]	268	[6-8]	270	[9-9]	273	[10-30]	274
f_7	1247	[0-0]	57	[1-1]	62	[2-2]	73	[3-7]	76										
f_8	1247	[0-0]	63	[1-1]	64	[2-2]	72	[3-6]	83										
f_9	1277	[0-0]	263	[1-1]	303	[2-2]	318	[3-3]	321	[4-4]	322	[5-5]	323	[6-26]	347				
f_{10}	1389	[0-0]	253	[1-1]	304	[2-2]	324	[3-3]	341	[4-4]	351	[5-5]	355	[6-7]	363	[8-9]	399	[10-10]	409
		[11-12]	415	[13-13]	419	[14-16]	429	[17-18]	431										
f_{11}	1183	[0-0]	79	[1-1]	113	[2-9]	114												
f_{12}	1204	[0-0]	58	[1-1]	67	[2-2]	75	[3-7]	76										
f_{13}	3844	[0-0]	470	[1-1]	504	[2-2]	528	[3-3]	533	[4-4]	534	[5-11]	650						
f_{14}	3832	[0-0]	333	[1-1]	365	[2-2]	386	[3-5]	391	[6-6]	442	[7-10]	446						
f_{15}	3854	[0-0]	428	[1-1]	453	[2-2]	495	[3-3]	499	[4-4]	503	[5-5]	560	[6-6]	574	[7-7]	576	[8-10]	577
f_{16}	3848	[0-0]	432	[1-1]	462	[2-2]	487	[3-3]	498	[4-4]	501	[5-5]	634	[6-6]	650	[7-13]	658		
f_{17}	3854	[0-0]	426	[1-1]	480	[2-2]	525	[3-3]	539	[4-4]	540	[5-5]	559	[6-11]	570				
f_{18}	3848	[0-0]	469	[1-1]	547	[2-2]	551	[3-3]	553	[4-4]	635	[5-5]	672	[6-10]	674				

Table 6.1: Lazy abstraction. N stands for the name of the verified property. ♯*Vars* stands for the number of state variables in the concrete model M_c. ♯TF stands for the time frames and ♯AV represents the number of variables (defining the abstract TR_i) in the abstract model M_i at the given time frame i (appearing in the column ♯TF).

N	♯Vars	Stat	♯V[Ω]	♯V[Ω_L]	♯C[Ω]	♯C[Ω_L]	k	k_L	T	T_L
f_1	11866	false	1001	**818**	8457	**3939**	15	18	1646	**599**
f_2	5693	true	236	**11**	617	**62**	14	8	133	**9.2**
f_3	5693	true	229	**121**	1314	**570**	13	8	351	**40.5**
f_4	5693	true	104	**24**	2101	**32**	32	14	513	**13.6**
f_5	5773	true	> 616*	**414**	> 16689*	**12425**	7*	35	TO	1223
f_6	1183	true	432	**370**	50511	**29316**	36	31	2216	2763
f_7	1247	true	250	**152**	10732	**238**	11	8	432	**2.6**
f_8	1247	true	177	**96**	14702	**293**	8	7	520	**3.5**
f_9	1277	false	357	**331**	8762	**3788**	13	27	164	**101**
f_{10}	1389	false	**397**	417	**12455**	19742	13	19	**262**	1268
f_{11}	1183	true	114	**106**	29183	**2589**	9	10	1153	**109**
f_{12}	1204	true	114	**105**	18698	**229**	8	8	818	**3.0**
f_{13}	3844	true	**320**	578	**547**	1529	10	12	**16.7**	59.1
f_{14}	3832	true	650	**488**	2414	**1553**	12	11	117	**61**
f_{15}	3854	true	> 470*	**666**	> 8320*	**5363**	6*	11	TO	**730**
f_{16}	3848	true	> 687*	**826**	> 7733*	**5506**	8*	14	TO	**381**
f_{17}	3854	true	811	**673**	10934	**1837**	13	12	919	**83**
f_{18}	3848	true	898	**716**	9889	**2080**	13	11	1891	**84**
f_{19}	3848	true	**966**	> 216*	**13370**	> 266*	11	7*	**2225**	TO

Table 6.2: Running parameters for various properties. N stands for the name of the verified property. ♯*Vars* stands for the number of state variables in the cone of influence. ♯V[Ω] - number of variables in Ω, ♯C[Ω] - number of clauses in Ω, k - size of $\Omega(M, p)$ and T - the runtime in seconds. The subscript L represents the value for the *Lazy* version (L-IC3).

Chapter 7

Conclusion

We presented four methods that aim at improving existing SAT-based unbounded model checking. The first three, ISB, DAR and CNF-ITP, are based on interpolation. With ISB and DAR we show different ways to use interpolants for reachability analysis. With CNF-ITP, we present a novel method for interpolants computation that yields interpolants in CNF. Further we show how this fact can be used in an interpolation-based algorithm. The last method we present, L-IC3, is based on IC3. Unlike interpolation-based methods, IC3 and L-IC3 are based on local reachability checks. With L-IC3 we show how to tightly integrate a *lazy abstraction* mechanism into IC3.

We believe there is more to be done in order to make SAT-based model checking more efficient. One possibility is to integrate interpolation and IC3-style approaches in a tighter manner. Each of the approach uses a different generalization mechanism when computing the over-approximation of reachable states. It may be the case, that integrating the two may yield better performance.

In the context of interpolation, another possibility is to better understand interpolants in the context of *resolution proofs*. By analyzing the mechanics of CDCL SAT-solvers, one can try and make the proof produced by such solvers better suited for interpolants in the context of model checking.

Bibliography

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In Craig Boutilier, editor, *IJCAI*, pages 399–404, 2009.

[2] Omer Bar-Ilan, Oded Fuhrmann, Shlomo Hoory, Ohad Shacham, and Ofer Strichman. Linear-time reductions of resolution proofs. In Hana Chockler and Alan J. Hu, editors, *Haifa Verification Conference*, volume 5394 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2008.

[3] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness checking as safety checking. *Electr. Notes Theor. Comput. Sci.*, 66(2):160–177, 2002.

[4] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[5] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In Rance Cleaveland, editor, *TACAS*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[6] 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*. FMCAD Inc., 2011.

[7] Aaron R. Bradley. k-step relative inductive generalization. *CoRR*, abs/1003.3649, 2010.

[8] Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

[9] Aaron R. Bradley and Zohar Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD*, pages 173–180. IEEE Computer Society, 2007.

[10] Aaron R. Bradley, Fabio Somenzi, Zyad Hassan, and Yan Zhang. An incremental approach to model checking progress properties. In Bjesse and Slobodová [6], pages 144–153.

[11] 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*. Springer, 2002.

[12] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[13] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[14] Gianpiero Cabodi, Marco Murciano, Sergio Nocco, and Stefano Quer. Stepping forward with interpolants in unbounded model checking. In Soha Hassoun, editor, *ICCAD*, pages 772–778. ACM, 2006.

[15] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Mixing forward and backward traversals in guided-prioritized bdd-based verification. In Brinksma and Larsen [11], pages 471–484.

[16] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Interpolation sequences revisited. In *DATE*, pages 316–322. IEEE, 2011.

[17] Hana Chockler, Alexander Ivrii, and Arie Matsliah. Computing interpolants without proofs. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Haifa Verification Conference*, volume 7857 of *Lecture Notes in Computer Science*, pages 72–85. Springer, 2012.

[18] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. SATABS: SAT-based predicate abstraction for ANSI-C. In Halbwachs and Zuck [34].

[19] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *JACM*, 2003.

[20] Edmund Clarke Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT press, 1999.

[21] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[22] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[23] W. Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *J. Symb. Log.*, 22(3), 1957.

[24] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[25] Vijay D'Silva, Daniel Kroening, Mitra Purandare, and Georg Weissenbacher. Interpolant strength. In *VMCAI*, pages 129–145, 2010.

[26] Vijay D'Silva, Mitra Purandare, and Daniel Kroening. Approximation refinement for interpolation-based model checking. In Francesco Logozzo, Doron Peled, and Lenore D. Zuck, editors, *VMCAI*, volume 4905 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2008.

[27] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[28] Niklas Eén, Alan Mishchenko, and Robert K. Brayton. Efficient implementation of property directed reachability. In Bjesse and Slobodová [6], pages 125–134.

[29] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[30] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *DATE*, pages 886–891, 2003.

[31] Orna Grumberg, Assaf Schuster, and Avi Yadgar. Memory efficient all-solutions sat solver and its application for reachability analysis. In Hu and Martin [37], pages 275–289.

[32] A. Gupta, M. Ganai, Z. Yang, and P. Ashar. Iterative abstraction using SAT-based BMC with proof analysis. In *ICCAD*, 2003.

[33] A. Gupta and O. Strichman. Abstraction refinement for bounded model checking. In *CAV*, 2005.

[34] Nicolas Halbwachs and Lenore D. Zuck, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*. Springer, 2005.

[35] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL*, 2004.

[36] T.A. Henzinger and R. Majumdar R. Jhala. Lazy abstraction. In *POPL*, 2002.

[37] Alan J. Hu and Andrew K. Martin, editors. *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, volume 3312 of *Lecture Notes in Computer Science*. Springer, 2004.

[38] Ranjit Jhala and Kenneth L. McMillan. Interpolant-based transition relation approximation. In *CAV*, 2005.

[39] Orna Kupferman and Moshe Y. Vardi. Model Checking of Safety Properties. In *CAV*, pages 172–183, 1999.

[40] R. P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton University Press, 1994.

[41] John Launchbury and John C. Mitchell, editors. *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*. ACM, 2002.

[42] João Marques-Silva. Interpolant learning and reuse in SAT-based model checking. *Electr. Notes Theor. Comput. Sci.*, 174(3):31–43, 2007.

[43] K. L. McMillan. Interpolation and SAT-based Model Checking. In *CAV*, 2003.

[44] K. L. McMillan. Lazy Abstraction with Interpolants. In *CAV*, 2006.

[45] K. L. McMillan and N. Amla. Automatic Abstraction without Counterexamples. In *TACAS*, 2003.

[46] Kenneth L. McMillan. Applying sat methods in unbounded symbolic model checking. In Brinksma and Larsen [11], pages 250–264.

[47] Kenneth L. McMillan. Applications of craig interpolants in model checking. In Halbwachs and Zuck [34], pages 1–12.

[48] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *DAC*, pages 530–535. ACM, 2001.

[49] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[50] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3), 1997.

[51] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[52] Simone Rollini, Roberto Bruttomesso, and Natasha Sharygina. An efficient and flexible approach to resolution proof reduction. In Sharon Barner, Ian G. Harris, Daniel Kroening, and Orna Raz, editors, *Haifa Verification Conference*, volume 6504 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2010.

[53] Vadim Ryvchin and Ofer Strichman. Faster extraction of high-level minimal unsatisfiable cores. In Karem A. Sakallah and Laurent Simon, editors, *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2011.

[54] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.

[55] João P. Marques Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[56] Christian Stangier and Thomas Sidle. Invariant checking combining forward and backward traversal. In Hu and Martin [37], pages 414–429.

[57] Yakir Vizel and Orna Grumberg. Interpolation-sequence based model checking. In *FMCAD*, pages 1–8. IEEE, 2009.

[58] Yakir Vizel, Orna Grumberg, and Sharon Shoham. Lazy abstraction and sat-based reachability in hardware model checking. In Gianpiero Cabodi and Satnam Singh, editors, *FMCAD*, pages 173–181. IEEE, 2012.

[59] Yakir Vizel, Orna Grumberg, and Sharon Shoham. Intertwined forward-backward reachability analysis using interpolants. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2013.

[60] Yakir Vizel, Vadim Ryvchin, and Alexander Nadel. Efficient generation of small interpolants in cnf. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 330–346. Springer, 2013.

[61] Lintao Zhang and Sharad Malik. Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In *SAT*, 2003.

[62] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885, 2003.

בפרק 3 אנו מציגים אלגוריתם מבוסס סדרת אינטרפולנטים אשר נקרא ISB. אלגוריתם זה מבוסס על BMC. בכל איטרצית BMC כאשר מחפשים אחר דוגמא נגדית, במידה וכזו אינה קיימת, סדרת אינטרפולנטים מחושבת כך שהיא מייצגת קירוב יתר של מצבים ישיגים במערכת. בשונה מהאלגוריתם הראשון המבוסס אינטרפולנטים, ISB לא מוסיף את האינטרפולנטים אל הנוסחה אותה בודק ה ... ובכך מבטל את השפעת גודל האינטרפולנט.

בפרק 4 אנו מציגים אלגוריתם שמחשב מצבים ישיגים גם מהמצבים ההתחלתיים (על ידי חישוב קידמי) ומצבים שיכולים להגיע אל המצבים הרעים (על ידי חישוב אחורי). אלגוריתם זה, שנקרא DAR, יכול להראות כמו גרסה משופרת של ISB. החישוב הקידמי והאחורי ב DAR מופעלים בצורה אחודה. כלומר, החישוב הקידמי משתמש במידע מן החישוב האחורי ולהיפך. בשונה מ ITP ומ ISB, DAR הוא ברובו מקומי. כלומר, הוא אינו דורש פריסה של רלציית המעברים ברוב המקרים.

בפרק 5 אנו מציגים אלגוריתם חדשני לחישוב אינטרפולנטים עבור בדיקת מודל. בשונה מאלגוריתמים קודמים שהשתמשו בחישוב אינטרפולנטים כללי, אנו מציגים חישוב המונחה על ידי הבעיה אותה מנסים לפתור. האלגוריתם פועל בשני שלבים. בשלב הראשון, מחושב קירוב של אינטרפולנט. קירוב זה מקיים רק חלק מתכונות האינטרפולנט ועל כן, בשלב השני, הופכים את הקירוב לאינטרפולנט על ידי תהליך שנקרא "חיזוק". החיזוק מתייחס למבנה הבעיה הספציפי בכך מתקבל אלגוריתם יעיל. תכונה חשובה של אלגוריתם זה הינה העובדה שהאינטרפולנט המחושב הינו בצורה נורמלית הנקראת Conjuctive Normal Form (CNF). גם בתכונה זו אנו עושים שימוש ומציגים גריסה משופרת של האלגוריתם ITP המשתמשת בעובדה שהאינטרפולנט נתון כ CNF.

בפרק 6 אנו מציגים גרסה חדשה של IC3 הנקראת LIC3. בגרסה זו נעשה שימוש במנגנון של אבסטרקציה עצלה. אך לא כמו אלגוריתמים אחרים בהם האבסטרקציה מנותקת מאלגוריתם האימות, LIC3, משלב את האבסטרקציה ואת IC3 לכדי אלגוריתם אחוד.

האבסטרקציה בה אנו משתמשים היא אבסטרקציה של משתנים נראים. כלומר, בכל שלב של החישוב אנו משתמשים בקבוצה שונה של משתנים מהמערכת הנתונה.

השימוש באבסטרקציה ב LIC3 גורם לאלגוריתם לבצע בדיקות SAT יעילות יותר וגם הופך את תהליך ההכללה, עליו IC3 נשען ליעיל יותר.

להעלות את רמת הדיוק. רמת הדיוק עולה כאשר מגדילים את אורך הדוגמאות הנגדיות אחריהן מחפשים. כפי שציינו, האורך נקבע על ידי הלולאה החיצונית, לכן הלולאה הפנימית עוצרת במקרה הנ"ל והלולאה החיצונית עוברת לאיטרציה הבאה.

קיימות שתי בעיות אותן אנחנו מנסים לפתור בשיטה זו. מאחר והאינטרפולנט הוא נוסחה פסוקית שמחליפה את המצבים ההתחלתיים, אם נוסחה זו היא מורכבת וגדולה, ייתכן ולא נוכל לפתור את בעייתה הספיקות. לכן, הבעיה הראשונה היא גודלם של האינטרפולנטים המחושבים. הבעיה השניה היא היא הצורך להגדיל את האורך של הדוגמאות הנגדיות אותן מחפשים על מנת להעלות את רמת הדיוק של קירוב היתר בו אנו משתמשים.

שיטה אלטרנטיבית, שכמו ITP, מבוססת ספיקות ומחשבת קירוב יתר של מצבים ישיגים הוצגה על ידי אהרון בראדלי ונקראת IC3. בשונה מ ITP, IC3 אינו משתמש בפריסה של רלציית המעברים על מנת להעלות את רמת הדיוק של הקירוב. IC3 משתמש בבדיקות ישיגות מקומיות אשר עושות שימוש במופע אחד של רלציית המעברים.

בעוד ITP משתמש ב SAT-solver כקופסא שחורה בזמן החיפוש אחר דוגמא נגדית ובזמן ההכללה, IC3 מנהל את החיפוש אחר הדוגמא הנגדית ואת ההכללה. IC3 מבוסס על חיפוש לאחור. כלומר, האלגוריתם מתחיל ממצב רע ומנסה לבנות דוגמא נגדית על ידי מציאת מצב קדמון שמגיע אל המצב הרע. לאחר מכן, מן המצב הקדמון, ינסה האלגוריתם למצוא מצב קדמון נוסף עד אשר לא יוכל להתקדם אחורה יותר או עד שיגיע למצבים ההתחלתיים ובכך ימצא דוגמא נגדית. במקרה ולא נמצאת דוגמא נגדית, IC3 מכליל את הסיפא שמצא לכדי מידע בנוגע למצבים היישיגים במערכת. ההכללה מתבצעת על ידי הליך שנקרא הכללה אינדוקטיבית (inductive generalization). תהליך זה הוא חלק ממה שעושה את IC3 יעיל.

תהליך ההכללה האינדוקטיבי אמנם עושה את IC3 יעיל, אך הוא גם נקודת החולשה של האלגוריתם. כאשר תהליך ההכללה נכשל, IC3 פועל בצורה שדומה לאינומרציית מצבים. כאשר מספר המצבים במערכת גדול, אינומרציית מצבים הינה בלתי יעילה ובמקרים אלה IC3 איננו יעיל.

הגישה שלנו לבידקת מודל מבוססת ספיקות

מטרת המחקר הינה למצוא מענה לנקודות החולשה שהזכרנו קודם לכן. בהקשר של אינטרפולציה, אנו מתמודדים עם גודל האינטרפולנטים או על ידי שימוש שונה מהם כחלק מאלגוריתם האימות או על ידי חישוב שונה של האינטרפולנט. בהקשר של IC3 אנו מוסיפים אלמנט של אבסטרקציה עצלה לאלגוריתם על מנת להפוך אותו ליעיל יותר.

כעת נסקור חלק מהאתגרים בבדיקת מודל מבוססת ספיקות. כמו כן, נסקור את הדרך שלנו להתמודדות עם אתגרים אלה.

האתגרים בבדיקת מודל מבוססת ספיקות

עבודה זו מוקדשת לשיפורה של בדיקת מודל מבוססת אינטרפולציה ומבוססת IC3/PDR. שיטות אלה מחשבות את קבוצות המצבים הישיגים במערכת על ידי שימוש בקירוב יתר. החישוב מתבצע תוך כדי ניסיון להראות כי לא קיימת דוגמא נגדית עבור התכונה הנבדקת. קירוב היתר של המצבים הישיגים מתבצע על ידי *הכללה*. הכללה הינה התהליך של הסקת מסקנה כללית מתוך מידע על מקרה פרטי.

בדיקת מודל מבוססת אינטרפולציה (ITP) הוצגה לראשונה על ידי קן מקמילן. בשיטה זו, קירוב היתר של המצבים הישיגים מחושב על ידי שימוש באינטרפולנטי קרייג (Craig interpolants). המשפט של קרייג מציין כי בהינתן זוג נוסחאות A ו B כך שהגימום של שתי הנוסחאות אינו ספיק, ניתן לחשב נוסחה שלישית I כך ש I היא נוסחה מעל ה א"ב המשותף ל A ו B, I נגררת מ A והגמום של I ו B אינו ספיק. האינטרפולנטים מחושבים מתוך הוכחת רזולוציה שמיוצרת על ידי SAT-solver כאשר נוסחה היא איננה ספיקה. הנוסחאות בהן ITP עושה שימוש הן נוסחאות BMC. כאשר נוסחת BMC היא בלתי ספיקה, ניתן להסיק כי לא קיימת דוגמא נגדית באורך מסויים. על ידי שימוש במשפט של קרייג ניתן לחשב אינטרפולנט שמייצג קירוב יתר של מצבים ישיגים. מאחר וניתן לכרות מידע על המצבים הישיגים, ניתן להשתמש בשיטה זו על מנת להוכיח נכונות של תכונה.

אינטרפולנטים הנם דוגמא להכללה. מאחר ופתרון בעיית ה BMC משמש ככלי להוכחת אי קיום של דוגמא נגדית באורך מסויים, היכולת להסיק מסקנה בנוגע לדוגמאות נגדיות ארוכות יותר הוא צורה של הכללה.

ITP עובד בצורה איטרטיבית ומבוסס על לולאה מכוננת. הלולאה החיצונית שולטת על אורך הדוגמא הנגדית שמחפשים ב BMC והלולאה הפנימית בודקת נוסחאות BMC באורך קבוע ומחשבת אינטרפולנטים כל עוד נוסחות אלה הן אינן ספיקות. כאשר נוסחת BMC היא בלתי ספיקה, מחושב אינטרפולנט. האינטרפולנט מחליף את המצבים ההתחלתיים מהם מתחיל החיפוש אחר דוגמא נגדית באיטרציה הבאה. מאחר והאינטרפולנטים מייצגים קירוב יתר של מצבים ישיגים, ניתן להשתמש בהם על מנת לקבוע מתי הושגה נקודת שבת. נקודת השבת מוכיחה כי כל המצבים הישיגים במערכת נבדקו וכולם מקיימים את התכונה ולכן ניתן להסיק כי המערכת מקיימת את התכונה הנבדקת. במצב שנוסחת BMC הנבדקת בלולאה הפנימית היא ספיקה, אם המצבים ההתחלתיים לא שונו (איטרציה ראשונה) ניתן להסיק כי קיימת דוגמא נגדית. אם לא, לא ניתן להסיק כי קיימת דוגמא נגדית ויודעים כי קירוב היתר המתבצע על ידי האינטרפולנטים הוא גס מידי ויש

תקציר

מערכות ממוחשבות חולשות כמעט על כל תחום של חיינו ופעולתם הנכונה היא הכרחית. *בדיקת מודל* הינה תהליך אימות אוטומטי שבהינתן מערכת בודק קיום של תכונה מסויימת ביחס למערכת זו. המערכת מתוארת בדרך כלל כמכונת מצבים בצורת גרף מעברים. התכונה נתונה כנוסחה בלוגיקה טמפורלית (Temporal Logic). בשונה מאימות מבוסס סימולציה, בדיקת מודל ממצאה מאחר והיא מכסה את כל ההתנהגויות האפשריות במערכת ומוכיחה את קיום התכונה או מוצאת דוגמא נגדית לאי קיום התכונה.

בדיקת מודל יושמה ומיושמת בבדיקת מערכות חומרה ותוכנה. אך המגבלה המרכזית של השיטה היא *בעיית התפוצצות המצבים*. בעיה זו מקורה במספר המצבים במערכות אמיתיות. בדיקת המודל דורשת זיכרון רב וחישוב מורכב ביחס לגודל המודל, ולכן, עבור מודלים גדולים, בדיקת מודל עלולה להיות בלתי יישימה. חלק נכבד מהמחקר בתחום מוקדש להתמודדות עם הבעיה.

השימוש בתרשימי החלטה בינריים (BDDs) היה הצעד המשמעותי הראשון בהתמודדות עם בעיית התפוצצות המצבים. בדיקת מודל סימבולית, מבוססת BDDs אפשרה בדיקת מודל של מערכות חומרה אמיתיות עם כמה מאות של משתני מצב. אך עדיין, חלקים ממערכות חומרה בעלי פונקציונליות מוגדרת מכילים אלפי משתני מצב ויותר. כדי להתמודד עם מערכות כאלה, פותחה *בדיקת מודל חסומה מבוססת ספיקות* (*BMC*). BMC מבוסס על פריסה של רצליית המעברים על מנת לחפש דוגמא נגדית באורך מסויים. אך למרות היכולת של שיטה זו להתמודד עם מערכות גדולות יותר, החסרון העיקרי שלה הוא היותה נאותה אך לא שלמה. כלומר, מטרתה העקרית היא מציאת דוגמאות נגדיות ולא הוכחת נכונות.

מספר שיטות פותחו על מנת להתמודד עם בעית השלמות של השיטה מבוססת הספיקות. *אינדוקציה, אינטרפולציה, סדרת אינטרפולציה, בנייה אינקרמנטלית של פסוקיות אינדוקטיביות לנכונות ודאית (IC3/PDR)* ו *IC3 עצל (L-IC3)* הן שיתות מבוססות ספיקות שהן נאותות ושלמות ויכולות להוכיח נכונות של תכונה עבור מערכת נתונה.

מתוך השיטות האלה L-IC3 משתמשת באבסטרקציה. אבסטרקציה היא שיטה נוספת להתמודדות עם בעיית התפוצצות המצבים. על ידי שימוש באבסטרקציה, חלקים מהמערכת, שאינם רלוונטיים להוכחת התכונה, מוסרים. על ידי הסרת החלקים הלא רלוונטיים, מתקבלת מערכת קטנה יותר המכילה פחות מצבים מה שמקל על בדיקת המודל. אבסטרקציה עצלה פותחה לראשונה עבור מערכות תוכנה. שיטה זו מאפשרת הסרת חלקים שונים של המערכת בשלבים שונים של האימות.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

בראש ובראשונה ברצוני להודות למנחה שלי, פרופ' אורנה גימברג. היא הובילה אותי בסבלנות במסעי כחוקר מתחיל, עזרה לי ככל האפשר אך גם איפשרה לי לגדול ולפתח את עצמאותי האקדמית. הייתה לי הזכות לעבוד עם חוקרת בעלמת שם עולמי בתחום האימות הפורמלי ואני בטוח שהיא נתנה לי את כל הכלים הדרושים על מנת להמשיך ולהצליח.

ברצוני להודות לד״ר שרון שוהם, ד״ר אלכס נאדל וד״ר ודים ריבקין על שיתוף פעולה פורה ומוצלח. אני בטוח שהניסיון שצברתי תוך כדי שיתוף הפעולה ילווה אותי בהמשך דרכי. כמו כן, ברצוני להודות לד״ר זיאד חנא על התמיכה וההזדמנות שנתן לי לממש את מחקרי בסביבה תעשייתית ובכך נתן לי את הכלים לראות כיצד מחקרי משפיע על בעיות מן העולם האמיתי.

ברצוני להודות להורי, ישראל וסימה ויזל, שלימדו אותי לשאול, לחקור, להתמיד ולהאמין בעצמי. הם תמכו בי בכל ההיבטים של לימודי, מבית הספר היסודי ועד בית הספר ללימודי מוסמכים. אני מודה גם לאחי ואחותי, שי ושיר, על שהביעו תמיכה והערכה לפועלי. כל אלה שימשו השראה למחקרי.

בדיקת מודל מבוססת ספיקות על ידי שימוש באינטרפולציה ובנייה אינקרמנטלית של פסוקיות אינדוקטיביות לנכונות ודאית

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר

דוקטור לפילוסופיה

יקיר ויזל

הוגש לסנט הטכניון מכון טכנולוגי לישראל

אייר תשע"ד חיפה מאי 2014

בדיקת מודל מבוססת ספיקות על ידי שימוש באינטרפולציה ובנייה אינקרמנטלית של פסוקיות אינדוקטיביות לנכונות ודאית

יקיר ויזל