Abstract Counterexamples for Non-disjunctive Abstractions

K. L. McMillan[1] and L. D. Zuck[2⋆]

[1] Cadence Research Labs
[2] University of Illinois at Chicago

Abstract. Counterexample-guided abstraction refinement (CEGAR) is an important method for tuning abstractions to properties to be verified. The method is commonly used, for example in selecting predicates for predicate abstraction. To date, however, it has been applied primarily to powerset abstractions, which allow one to speak of an abstract transition system and abstract states. Here, we describe a general framework for CEGAR in non-disjunctive abstractions by introducing a generalized notion of abstract counterexample, and methods for computing such counterexamples. We apply this framework to Indexed Predicate Abstraction (IPA), a promising technique for synthesizing quantified inductive invariants of infinite-state systems. In principle, it can be applied to other non-disjunctive abstractions occurring in program analysis.

1 Introduction

Effective application of abstract interpretation depends on choosing the right abstract domain. This domain must be rich enough to contain an inductive invariant that proves a given property, but not so rich as to make analysis intractable. One very fruitful approach to choosing abstractions has been abstraction refinement. That is, when our abstract domain fails to prove a given property, we analyze this failure, producing a refined abstract domain that rules out some class of failures. This process repeats until either the property is proved, or analysis reveals that the property is false, or computational resources are exhausted. A particularly successful form of abstraction refinement is *counterexample-guided abstraction refinement*, or CEGAR [10, 2]. In this approach, when the abstract domain fails to prove the property, we produce an *abstract counterexample*. This is a sequence of abstract states in which every transition is allowed by the abstract transformer, and the property is violated. An abstract state is, in effect, an atom of the abstract lattice. We refine the abstract domain so as to rule out the abstract counterexample. Abstract counterexamples both focus and simplify the refinement process, since they allow us to consider a limited class of behaviors.

⋆ This material was based on work supported by the National Science Foundation, while Lenore Zuck was working at the Foundation. Any opinion, finding, and conclusions or recommendations expressed in this article are those of the author and do not necessarily reflect the views of the National Science Foundation.

CEGAR has been applied effectively to a variety of domains, including localization abstractions [10] and predicate abstraction [18]. Its use is limited, however, by the fact that it applies only to abstract domains that are *disjunctive* (*i.e.*, closed under union). It is this condition that allows us to construct abstract counterexamples. For example, CEGAR cannot be applied directly to indexed predicate abstraction [11] (IPA) because this abstraction is not disjunctive.

In this work, we generalize the notion of abstract counterexample to a construct we call a *minimal sufficient explanation* (MSE). An MSE is a sequence of elements of the abstract domain that may not be atoms. In the case of an abstract lattice that is atomistic and disjunctive, however, it reduces to the standard notion of abstract counterexample. An MSE may be used to focus abstraction refinement in much the same way as an abstract counterexample, for example, using the interpolation approach [7]. Our primary motivation in this work is to be able to effectively refine indexed predicate abstractions, and we will use this method as an example application.

Related work Existing work on refinement of non-disjunctive abstractions is not based on abstract counterexamples. Typically, the weakest liberal precondition operator is iterated. This allows us to find the first point in the abstract fixed point series in which lost information resulted in inclusion of a bad concrete state (one reaching a state violating the property). The abstraction is refined at this point. For example, Gulavani and Rajamani do this by eliminating widenings at specific points in the fixed point series [6].

In the terminology of this paper, the sequence of (negations of) the weakest preconditions of the property is a *sufficient explanation* for the failure to prove the property. However, it is not *minimal* with respect to the given abstract domain, nor is it generally even expressible in that domain. Using a *minimal* sufficient explanation allows us to focus on a restricted set of concrete behaviors. In this way, we hope to gain both efficiency and better focus on relevant refinements, as in CEGAR. Moreover, this avoids having to deal with the series of weakest preconditions, which may have deeply nested quantifiers in the case of programs with input or non-deterministic choice.

Since one of the goals of this work is to produce quantified inductive invariants, we mention some other work in this area. Lahiri presents a collection of heuristics based on the weakest precondition operator for guessing indexed predicates [11], but leaves open the question of how to apply CEGAR. Henzinger, *et al.*, use interpolants for predicate refinement [7], but without index variables. The method of *invisible invariants* [16] can effectively synthesize quantified invariants, but only for families of finite-state systems. IPA can also handle infinite-state systems.

Outline The paper is organized as follows. Section 2 introduces the notion of MSE, or generalized abstract counterexample. Section 3 then reviews the method of indexed predicate abstraction, and shows how to compute MSE's for this application. In section 4, we show how indexed predicates can be derived from interpolants, and how, in principle, MSE's can be used to drive this process in a CEGAR loop.

2 Generalized abstract counterexamples

In this section, we generalize the concept of abstract counterexample. The idea is to view an abstract counterexample not as a run of an abstract transition system, but rather as a minimal sufficient explanation of the failure of the abstraction to prove a given property. We will see that in the case of powerset abstractions, these two notions coincide.

Abstract interpretation First we review some concepts from abstract interpretation [3]. Consider a concrete transition system with set of states S, initial states $I \subseteq S$, and transition relation $T \subseteq S \times S$. We can define a concrete transformer $\tau(s) = I \cup T(s)$, where $T(s)$ is the image of s with respect to T. The least fixed point of τ is the set of concrete reachable states of the system.

An *abstraction* of the system is defined by an abstract lattice L and a monotone concretization function $\gamma : L \rightarrow S$. The abstract lattice is ordered by $\sqsubseteq$, with least upper bound operator $\sqcup$ and greatest lower bound operator $\sqcap$, usually referred to as "join" and "meet" respectively. If we think of L as a logical language, then γ defines the semantics of the language, with $\gamma(p)$ giving the extension of predicate $p \in L$.

We will assume that L is finite and intersection-closed, that is, for any $p, q \in L$, there exists $r \in L$ such that $\gamma(r) = \gamma(p) \cap \gamma(q)$. In this case, γ is the upper adjoint of a Galois connection, whose lower adjoint is:

$$\alpha(s) = \sqcap\{p \mid s \subseteq \gamma(p)\}$$

The abstraction function α gives the best abstract approximation of a set of states s, which can be thought of as the conjunction of all the predicates in L that are valid over s.

This in turn gives us a *best abstract transformer*, $\tau^\sharp = \alpha \circ \tau \circ \gamma$. For any predicate $p \in L$, this function yields the best abstract approximation of the set of successors of states in p. The fixed points of $\tau^\sharp$ are all the inductive invariants in L, and the least fixed-point is the strongest of these. Thus, to prove that a given set $F \subseteq S$ is unreachable, we compute the least fixed point of $\tau^\sharp$, as the stable limit of the series $(\tau^\sharp)^i(\bot)$. Then, if $\mathrm{lfp}(\tau^\sharp) \sqcap \alpha(F) = \bot$ we say the abstraction proves F unreachable. On the other hand, if $(\tau^\sharp)^i(\bot) \sqcap \alpha(F) \neq \bot$ for any $i > 0$, then the abstraction fails to prove unreachability of F.

Explanation of failures What then would constitute a minimal sufficient explanation for such a failure? Consider first the case of a single transition. Given two predicates $p, q \in L$, we will say that p is a *minimal sufficient precondition* (MSP) of q when $\tau^\sharp(p) \sqsupseteq q$ and there is no $\dot{p} \sqsubset p$ such that $\tau^\sharp(\dot{p}) \sqsupseteq q$. That is, p is a minimal element of the abstract lattice sufficient to guarantee at least q

at the next time. Put another way, p is an explanation of why the abstraction produced q at the next time.

Now we extend this notion to a reachability computation. We will say that a sequence $x_0, \ldots x_k \in L^*$ is a *minimal sufficient explanation* (MSE) for failure to prove unreachability of F, when it is pointwise minimal such that:

- $x_0 = \bot$ and
- for all $0 \leq i < k$, $\tau^\sharp(x_i) \sqsupseteq x_{i+1}$, and
- $x_k \sqcap \alpha(F) \neq \bot$

That is, each element of the sequence is a MSP of its successor, and the last element fails to rule out F.

The notion of MSE corresponds precisely to the notion of "abstract counterexample" in the traditional CEGAR framework. This framework applies only to *powerset abstractions*. This means that γ is disjunctive (join-preserving) in the sense that $\gamma(p \sqcup q) = \gamma(p) \cup \gamma(q)$. Moreover, it requires that L be *atomistic*, in that every element is the join of some set of atoms (elements that cover $\bot$). For example, in predicate abstraction, the join operation is logical disjunction (*i.e.*, union over sets of states) and the atoms are the minterms over the abstraction predicates P (a minterm over P is a conjunction of literals over P in which each predicate in P occurs once).

In this case, we can think of the atoms of L as "states" of an abstract transition system. That is, because of the disjunctive join and atomicity, $\tau^\sharp$ is pointwise over atoms:

$$\tau^\sharp(p) = \sqcup\{\tau^\sharp(a) \mid a \in \text{atoms}(p)\} \sqcup \tau^\sharp(\bot)$$

It follows that an MSP of any atom is an atom or $\bot$. That is, if $\tau^\sharp(p) \sqsupseteq q$ and $p \neq \bot$, then p contains some atom a such that $\tau^\sharp(a) \sqsupseteq q$. Thus, in a disjunctive, atomistic abstraction such as predicate abstraction, MSE's contain only atoms, and we can think of them as sequences of abstract "states".

This is not true in the general case, however. For example, indexed predicate abstraction is atomistic but not disjunctive. As a result, an MSE is a sequence of *sets* of atoms. One way to view the occurrence of multiple atoms at some point in the MSE is that the abstract interpretation has lost information in merging multiple execution paths. Thus, no one path is sufficient to "explain" the successor state.

Computing generalized counterexamples Now we consider the problem of computing an MSE for failure to prove unreachability of F. Suppose that we have computed a sequence of fixed point approximations $x_i = (\tau^\sharp)^i(\bot)$ for $i = 0 \ldots k$ such that $x_k \sqcap \alpha(F) \neq \bot$. There is a simple but inefficient backward approach to computing an MSE. We start by setting x_k to any atom in $x_k \sqcap \alpha(F)$. Then for $i = k-1$ down to 1, we greedily reduce x_i in the lattice order so long as $\tau^\sharp(x_i) \sqsupseteq x_{i+1}$. If L is atomistic, this means greedily removing atoms from x_i. When x_i cannot be further reduced, it is an MSP for x_{i+1}, and we move on to x_{i-1}. At the end of this process, we have an MSE for the failure.

This simple approach could be computationally costly, because the height of the abstract lattice is typically exponential in some parameter of the abstraction (for example, in predicate abstraction it is exponential in the number of abstraction predicates). Thus, the number of reduction steps can also be exponential. To avoid this, we need some way of putting an upper on the MSP so that the number of reduction steps necessary to reach the MSP is also bounded. We will show how to do this in some special cases of practical interest.

Our basic problem is to compute a MSP for predicate q that is dominated by some predicate p. For any monotone transformer τ, we will write the set of sufficient preconditions of q dominated by p as:

$$\mathrm{SP}(p, \tau, q) = \{\hat{p} \mid \hat{p} \sqsubseteq p \text{ and } \tau(\hat{p}) \sqsupseteq q\}$$

The set of minima of this set will be denoted $\mathrm{MSP}(p, \tau, q)$. Our general approach will be to compute a SP, then iteratively remove atoms until it becomes a MSP.

To do this, we can rely on several useful properties of SP. First, SP is join-preserving, that is, if $\hat{p}_i \in \mathrm{SP}(p, \tau, q_i)$ then $\sqcup_i \hat{p}_i \in SP(p, \tau, \sqcup_i q_i)$. This is due simply to monotonicity of τ. It means that to compute a SP for q, we can simply take the join of SP's for the individual atoms of q. As we observed above, when τ is join-preserving, we need only consider SP's for atoms that are atoms. In addition, we can make use of the following results:

Theorem 1. *If τ is meet-preserving, then $SP(p, \tau, q)$ is closed under meets. Moreover, if $SP(p, \tau, q)$ is non-empty, the unique element of $MSP(p, \tau, q)$ is $\sqcap SP(p, \tau, q)$.*

Theorem 2 (Meet rule). *Let $\tau(s) = \sqcap_i \tau_i(s)$, for τ_i meet-preserving, and suppose $\hat{p}_i \in MSP(p, \tau_i, q)$. Then $\sqcup \hat{p}_i \in MSP(p, \tau, q)$.*

Theorem 3 (Chain rule). *If $\tau = \tau_2 \circ \tau_1$ and $\hat{p} \in MSP(p, \tau, r)$, then there exists $\hat{q}$ such that $\hat{p} \in MSP(p, \tau_1, \hat{q})$ and $\hat{q} \in MSP(\tau_1(p), \tau_2, r)$.*

The chain rule allows us to compute MSP's for a composition of transformers by working backward. We can think of the MSE computation as being one long application of this rule.

3 Indexed predicate abstraction

We now apply the notion of MSE to the problem of abstraction refinement for indexed predicate abstraction. We will apply IPA to transition systems represented symbolically using first-order logic. We break the abstract transformer for IPA into a composition of a join-preserving and a meet-preserving transformer. Then we use the chain rule and the meet rule to compute MSP's. By this means, we use a number of decision procedure calls which is quadratic in the final number of atoms in the MSE.

Symbolic transition systems Let Σ be a first-order signature consisting of individual variables and uninterpreted n-ary functional and propositional constants. A *state formula* is a first-order formula over Σ, (which may include various interpreted symbols, such as $=$ and $+$). We can think of a state formula ϕ as representing a set of states, namely, the set of first-order models of ϕ. We will express the proposition that an interpretation σ over Σ models ϕ by $\phi[\sigma]$, or $\sigma \models \phi$. If s is a set of interpretations, we will write $s \models \phi$ to mean that every element of s models ϕ.

We also assume a first-order signature Σ', disjoint from Σ, and containing for every symbol $v \in \Sigma$, a unique symbol v' of the same type. For any formula or term ϕ over Σ, we write ϕ' for the result of replacing every occurrence of a symbol v in ϕ with v'. Similarly, for any interpretation σ over Σ, we will denote by σ' the interpretation over Σ' such that $\sigma' v' = \sigma v$. A *transition formula* is a first-order formula over $\Sigma \cup \Sigma'$. We think of a transition formula T as representing a set of state pairs, namely the set of pairs (σ_1, σ_2), such that $\sigma_1 \cup \sigma_2'$ models T. We will express the proposition that $\sigma_1 \cup \sigma_2'$ models T by $T[\sigma_1, \sigma_2]$.

A *symbolic transition system* is a pair (I, T), where I is a state formula and T is a transition formula. We interpret this as a transition system whose initial states are represented by I and whose transition relation is represented by T.

Indexed predicate abstraction Indexed predicate abstraction [11] is similar to predicate abstraction, except that the predicates contain free variables that are implicitly universally quantified. We start with a distinguished set $J = \{i, j, k, \ldots\}$ of individual variables called the *index variables*, not occurring in I or T, and a finite set P of atomic formulas (possibly containing index variables). Our abstract lattice is the lattice L_P of Boolean combinations over P. In this lattice, the atoms are the minterms over P, which we can think of as either truth assignments to P, or conjunctions of literals over P. To avoid confusion between an *atom* of the abstract lattice and an *atomic* predicate, from here on we will refer to the lattice atoms as *minterms*.

Each element in L_P is a set of minterms. The lattice order $\sqsubseteq$ is set inclusion, and the meet and join are intersection and union, respectively. Alternately, we can think of meet and join as propositional conjunction and disjunction, and the lattice order as propositional implication (*i.e.*, where the propositions in P are uninterpreted). The concretization function is defined by:

$$\gamma(p) = \{\sigma \mid \sigma \models \forall J.\ p\}$$

That is, p represents the set of concrete states that model p for all valuations of the index variables. Because universal quantification distributes over conjunction, γ is meet-preserving. Thus the corresponding abstraction function is, by definition:

$$\alpha(s) = \sqcap\{p \mid s \models \forall J.\ p\}$$

which is equivalent to:

$$\alpha(s) = \{m \in \text{minterms}(P) \mid \sigma \models \exists J.\ m \text{ for some } \sigma \in s\}$$

This identity allows us to write the best abstract transformer as

$$\begin{aligned}\tau^\sharp(p) &= \{m \in \text{minterms}(P) \mid \sigma \models (\exists J.\ m) \text{ for some } \sigma \in \tau(\gamma(p))\}\\ &= \{m \in \text{minterms}(P) \mid (\forall J.\ p) \wedge T \wedge m' \text{ is sat.}\} \sqcup \alpha(I)\end{aligned}$$

That is, computing $\tau^\sharp(p)$ amounts to deciding $2^{|P|}$ satisfiability problems in first-order logic, one for each minterm over P. However, since first-order logic is undecidable, we make a further over-approximation by heuristically choosing a finite set of instantiations for the quantifiers.[3]

A *substitution* ρ for a set W of individual variables is a function that maps each variable in W to a first-order term. We will write $\rho(\phi)$ for the application of substitution ρ to formula ϕ (*i.e.*, the simultaneous replacement of each occurrence of variable $w \in W$ by $\rho(w)$). Given a set of substitutions $\mathcal{I}$, we write $\mathcal{I}(\phi)$ to denote $\bigwedge\{\rho(\phi) \mid \rho \in \mathcal{I}\}$. Note that if $\mathcal{I}$ is a set of substitutions for J, then $\forall J : \phi$ implies $\mathcal{I}(\phi)$.

Now we choose a finite set of substitutions $\mathcal{I}$ for the index variables and a suitable instantiation $\dot{T}$ of transition formula T. We might use, for example, the quantifier instantiation heuristics used in provers such as Simplify [5] for this purpose. The problem of instantiation is inherent in IPA, and not specifically related to abstraction refinement. The incompleteness of instantiation heuristics results in over-approximation. We can express the over-approximation of the best transformer as:

$$\dot{\tau}^\sharp(p) = \{m \in \text{minterms}(P) \mid \mathcal{I}(p) \wedge \dot{T} \wedge m' \text{ is sat.}\} \sqcup \alpha(I)$$

After instantiation, the satisfiability problems are quantifier-free, which means we can use an appropriate decision procedure, or reduce the problem to Boolean satisfiability. In [11], ALL-SAT methods are used to efficiently compute all the satisfying minterms. Even with these methods, $\dot{\tau}^\sharp$ may still be costly to compute in practice, so a weaker approximation may be called for.

Abstract counterexamples for IPA Notice that indexed predicate abstraction is not disjunctive. This is because universal quantification does not distribute over disjunction. In general, if p and q are two predicates in the abstract lattice, the concretization of their disjunction $\forall J.(p \vee q)$ is not equivalent to the disjunction of their concretizations $(\forall J.\ p) \vee (\forall J.\ q)$. As an example of this, suppose that a system chooses arbitrarily two process indices x and y, and transitions to a particular control state s when process x is in state p, but process y is not in state p. If we start from the minterm $p(i) \wedge \neg s$, then clearly we cannot transition to state s, since this represents $\forall i :\ p(i) \wedge \neg s$, states in which all processes are in state p. Similarly, if we start from the minterm $\neg p(i) \wedge \neg s$, we also cannot transition to s. However, if we start from disjunction $(p(i) \wedge \neg s) \vee (\neg p(i) \wedge \neg s)$,

[3] We could, of course, restrict ourselves to a decidable fragment, such as Presburger arithmetic, but this would not allow us to model, for example, parametrized protocols, or programs with unbounded arrays.

then we *can* transition to s. The abstract transformer is not point-wise over minterms, in the sense that a pair of minterms can have a successor that neither individual minterm has. For this reason, IPA does not yield abstract counterexamples.

However, we *can* compute MSE's and use these as an aid in refining the abstraction. We first observe that the computation of $\dot{\tau}^\sharp(p)$ can be broken into a sequence of two transformers: the instantiation of the index variables, followed by a forward image operation. The first transformer is function $\eta : L_P \to L_{\dot{P}}$, where $\dot{P} = \{\rho(\phi) \mid \phi \in P, \rho \in \mathcal{I}\}$, such that $\eta(p) = \mathcal{I}(p)$.

As an example, suppose that $J = \{i\}$, $P = \{s, p(i)\}$ and $\mathcal{I} = \{\rho_1, \rho_2\}$, where $\rho_1(i) = x$ and $\rho_2(i) = y$. Then $\dot{P} = \{s, p(x), p(y)\}$, and $\eta(p(i) \wedge \neg s) = p(x) \wedge p(y) \wedge \neg s$.

The second transformer is function $\delta : L_{\dot{P}} \to L_P$ that computes the predicate image with respect to $\dot{T}$. That is, let

$$\delta(p) = \{q \in \text{minterms}(P) \mid p \wedge \dot{T} \wedge q' \text{ is sat.}\}$$

This gives us $\dot{\tau}^\sharp(p) = \delta(\eta(p)) \sqcup \alpha(I)$. We show how to compute MSP's for δ and η, then combine these steps using the chain rule to compute MSP's for $\dot{\tau}^\sharp$.

Since δ is pointwise, the MSP's for a minterm q with respect to δ are also minterms. We can write the set of MSP's of minterm q as:

$$\begin{aligned} \text{MSP}(p, \delta, q) &= \{m \in \text{minterms}(\dot{P}) \mid m \sqsubseteq p \text{ and } \delta(m) \sqsupseteq q\} \\ &= \{m \in \text{minterms}(\dot{P}) \mid m \wedge p \wedge \dot{T} \wedge q' \text{ is sat.}\} \end{aligned}$$

Thus, finding one MSP for a minterm is a satisfiability problem. Because SP is join-preserving, we can compute a SP for any predicate q as the join of MSP's for the minterms of q. This SP can then be reduced to a MSP. Testing whether one minterm can be removed from the SP requires $|q|$ satisfiability tests (the number of minterms in q). Thus, in the worst case, the number of tests needed to compute an MSP is quadratic in $|q|$.

The transformer η is a meet over a finite set of transformers, that is, the individual substitutions:

$$\eta(p) = \sqcap\{\rho(p) \mid \rho \in \mathcal{I}\}$$

Thus, we can apply the meet rule, computing a MSP of η as a join over MPS's of the individual substitutions. Moreover, since substitutions are meet preserving, their MSP's are unique. Given a substitution ρ and a minterm $m \in L_{\dot{P}}$, there is a unique minterm $n \in L_P$ such that $\rho(n) \sqsupseteq m$. This is the one minterm such that for every literal l occurring in n, $\rho(l)$ occurs in m. Put another way, if we think of a minterm over P as a truth assignment to the predicates in P, then for all predicates $\phi \in P$, $n(\phi) = m(\rho(\phi))$.

Continuing the previous example, suppose that m is the minterm $s \wedge p(x) \wedge \neg p(y)$. Then the unique MSP of m with respect to ρ_1 is the minterm $s \wedge p(i)$. This is because m contains $\rho_1(s) = s$ and $\rho_1(p(i)) = p(x)$. In general, if m is a minterm in $L_{\dot{P}}$, we have:

$$\text{MSP}(p, \eta, m) = \{\lambda\phi.\ m(\rho(\phi))\}$$

Algorithm 1
Input: A pair $p, q \in L_P$
Output: An MSP $\hat{p}$ of q, such that $\hat{p} \sqsubseteq p$

```
1) Let $M = \emptyset$
2) For each minterm $m$ in $q \setminus \alpha(I)$:
3)        let $\hat{m}$ be a minterm over $\dot{P}$ s.t. $\mathcal{I}(p) \wedge \hat{m} \wedge \dot{T} \wedge m'$ is sat.
4)        add $\hat{m}$ to $M$
5) Greedily remove minterms from $M$, while $\delta(M) \sqsupseteq q$
6) For each minterm $m_i \in M$:
7)        let $n_i = \sqcup\{\{\lambda\phi.\ m_i(\rho(\phi)\} \mid \rho \in \mathcal{I}\}$
8) Return $\sqcup_i n_i$
```

Fig. 1. MSP computation for IPA

By the meet rule and the join-preserving property, the unique MSP for a predicate q with respect to η is:

$$\mathrm{MSP}(p, \eta, q) = \{\sqcup \cup \{\mathrm{MSP}(p, \rho, m) \mid \rho \in \mathcal{I},\ m \in \mathrm{minterms}(q)\}\}$$

Continuing our previous example, if $m = s \wedge p(x) \wedge \neg p(y)$ and $p = \top$, then $\mathrm{MSP}(p, \eta, m) = (s \wedge p(i)) \vee (s \wedge \neg p(i)) = s$. The first disjunct derives from ρ_1 and the second from ρ_2. This is a case where no single minterm serves as an MSP for a minterm.

With this result, we can now compute an MSP for the composition $\delta \circ \eta$ using the chain rule. To find an element of $\mathrm{MSP}(p, \delta \circ \eta, r)$, we first compute $q = \eta(p)$. Then let $\hat{q}$ be an element of $\mathrm{MSP}(q, \delta, r)$, and finally find an element of $\mathrm{MSP}(p, \eta, \hat{q})$. The resulting algorithm for computing a MSP in indexed predicate abstraction is shown in Figure 1.

In lines 1–4, we compute a SP of q with respect to δ, restricted to $\eta(p)$. At line 5, this is reduced to an MSP. In lines 6–8, we then compute the unique MSP with respect to η.

Notice that the number of satisfiability tests at line 3 is just $|q|$, the number of minterms in q. Each test of $\delta(M) \sqsupseteq q$ at line 5 could cost $|q|$ satisfiability tests, making the total number of tests quadratic in $|q|$. Alternatively, the test of $\delta(M) \sqsupseteq q$ might be done with a BDD image computation. The total number of decision problems we encounter is quadratic in the largest element of the MSE and linear in its length. Of course, this does not mean the number of minterms in the MSE cannot be exponential in $|P|$. For example, the number of minterms might double at each backward step.

4 Indexed predicates from interpolants

Interpolation has been used to derive relevant predicates for ordinary predicate abstraction from the refutation of counterexamples [7, 8]. This is one possible approach to counterexample-guided abstraction refinement. In this section, we extend this CEGAR technique to indexed predicate abstraction, using the notion of MSE introduced above. In effect, we make use of MSE's as constraints to simplify and focus the interpolant computation process.

Bounded model checking For any symbol s, and natural number i, we will use the notation $s^{\langle i \rangle}$ to represent the symbol s with i primes added. Thus, $s^{\langle 3 \rangle}$ is s'''. A symbol with i primes will be used to represent the value of that symbol at time i. We also extend this notation to formulas. Thus, the formula $\phi^{\langle i \rangle}$ is the result of adding i primes to every uninterpreted symbol in ϕ.

Now, given a system (I, T), we define a symbolic transformer formula $\mathcal{T} = T \vee I'$. This is defined so that the image of a set of states s with respect to $\mathcal{T}$ is exactly $\tau(s)$. The following formula is satisfiable exactly when $\tau^k(\bot) \cap F \neq \emptyset$, that is, when a state in F is reachable in $k-1$ steps or fewer from a state in I:

$$I^{\langle 1 \rangle} \wedge \mathcal{T}^{\langle 1 \rangle} \wedge \cdots \mathcal{T}^{\langle k-1 \rangle} \wedge F^{\langle k \rangle}$$

We will refer to this as a *bounded model checking* formula [1], since by testing satisfiability of such formulas, we can determine the reachability of a given condition within a bounded number of steps.

Interpolants from proofs Given a pair of formulas (A, B), such that $A \wedge B$ is inconsistent, an *interpolant* for (A, B) is a formula $\hat{A}$ with the following properties:

- A implies $\hat{A}$,
- $\hat{A} \wedge B$ is unsatisfiable, and
- $\hat{A}$ refers only to the common symbols of A and B.

Here, "symbols" excludes symbols such as $\wedge$ and $=$ that are part of the logic itself. Craig showed that for first-order formulas, an interpolant always exists for inconsistent formulas [4]. Of more practical interest is that, for certain proof systems, an interpolant can be derived from a refutation of $A \wedge B$ in linear time. For example, a purely propositional refutation of $A \wedge B$ using the resolution rule can be translated to an interpolant in the form of a Boolean circuit having the same structure as the proof [9, 17].

In [13] it is shown that linear-size interpolants can be derived from refutations in a first-order theory with uninterpreted function symbols and linear arithmetic. This translation has the property that whenever A and B are quantifier-free, the derived interpolant $\hat{A}$ is also quantifier-free. In [14], a method is described for computing universally quantified interpolants in first-order logic with equality, when such interpolants exist. In the sequel, we will assume that interpolants are universally quantified, and that the quantified variables are always drawn from J, the index set.

Heuristically, the chief advantage of interpolants derived from refutations is that they capture the facts that the prover derived about A in showing that A is inconsistent with B. Thus, if the prover tends to ignore irrelevant facts and focus on relevant ones, we can think of interpolation as a way of filtering out irrelevant information from A. Thus, atomic predicates occurring in the interpolant may be considered "relevant" to the refutation.

We can generalize the notion of interpolant to sequences of formulas. That is, given a sequence of formulas $\Gamma = \Gamma_1, \ldots, \Gamma_n$, we say that $A_0, \ldots A_n$ is an *interpolant* for Γ when

- $A_0 = \textsc{True}$ and $A_n = \textsc{False}$ and,
- for all $1 \leq i \leq n$, $A_{i-1} \wedge \Gamma_i$ implies A_i and
- for all $1 \leq i < n$, A_i refers only to common symbols between the prefix $\Gamma_1 \ldots \Gamma_i$ and the suffix $\Gamma_{i+1} \ldots \Gamma_n$.

An interpolant for a sequence can also be derived from a refutation of its constituent formulas.

We can use this concept to derive new indexed predicates sufficient to rule out a given abstract counterexample. In what follows, we will use a subscript to indicate the predicate set used to obtain a given quantity. Thus $\tau^\sharp_P$ is the abstract transformer obtained with predicate set P and so on. Now, suppose that using indexed predicate abstraction with predicates P, we obtain an MSE $X_P = x_0, \ldots, x_n$. The concretizations of X_P are all the sequences of concrete states $s_1, \ldots, s_n$ such that each s_i models $\gamma(x_i)$. If any such concretization is actually a failing run of (I, T), then the property is false. Otherwise, we would like to refine P by adding a set of predicates β sufficient to rule out X_P.

To this end, let Γ, the concretization sequence, be the following sequence of formulas:

$$I^{<1>}, (\mathcal{T} \wedge \gamma(x_1))^{<1>}, (\mathcal{T} \wedge \gamma(x_2))^{<2>}, \ldots, (\mathcal{T} \wedge \gamma(x_{n-1}))^{<n-1>}, (F \wedge x_n)^{<n>}$$

The conjunction of these formulas, $\wedge\Gamma$ is just a bounded model checking formula for (I, T) with the added constraint $\gamma(x_i)$ at each time i. The models of this conjunction are precisely the concretizations of X_P that are failing runs of (I, T). If we can prove $\wedge\Gamma$ unsatisfiable, then X_P is a false counterexample. We can then extract from the proof an interpolant as a sequence of quantified formulas for the form $\textsc{True}, A_1^{<1>}, \ldots, A_n^{<n>}, \textsc{False}$. We assume these are universal formulas, such that each $A_i = \forall J.\phi_{A_i}$. We now let β be the set of atomic predicates occurring in ϕ_{A_i} for any i. These are the predicates we will add to P in the next iteration of the refinement loop.

We can show that these predicates rule out future abstract counterexamples consistent with X_P. This is because the interpolant properties guarantee that $\tau^\sharp_\beta(A_i \wedge x_i) \Rightarrow A_{i+1}$. Thus, if X_P were a sufficient explanation for $\tau^\sharp_\beta$, then by induction we could show $x_i \Rightarrow A_i$, which implies that the interpolant sequence cannot end in $\textsc{False}$. This also implies that we cannot have $\beta \subseteq P$, since in this case X_p must also be a sufficient explanation for $\tau^\sharp_\beta$. Thus, the refinement step is guaranteed to add at least one new predicate. The practical function of the abstract counterexample is to act as a constraint on the bounded model checking problem, thus helping to make the refutation tractable.

We should note that using the method of [14] to generate interpolants has some limitations. The logic supported in that work is first order logic with equality. If other theories are needed, such as the theory of arrays or arithmetic, these must be axiomatized. This is necessarily incomplete for theories that have no finite axiomatization and can also be inefficient. In [14], however, it is shown that interpolation can successfully find invariants for simple heap manipulating programs, using simple arithmetic and an array theory with reachability predicates. Thus, there is some reason to think it may be effective for the more restricted task of abstraction refinement. The method described here can benefit from any future advances in interpolation methods. Moreover, the use of MSE's in abstraction refinement is not limited to interpolation methods, or for that matter to IPA. It might be applied to a variety of program analyses with non-disjunctive joins.

5 Conclusion

We have defined a notion of abstract counterexample for non-disjunctive abstract domains, called a minimal sufficient explanation. The notion of MSE reduces to the traditional notion of abstract counterexample for powerset abstractions. We showed how to compute MSE's for a particular example of a non-disjunctive abstraction, indexed predicate abstraction. The purpose of an abstract counterexample in the CEGAR framework is to simplify and focus the abstraction refinement process. We showed how this could be done with MSE's, using an interpolant-based refinement approach. In particular, we saw that universally quantified interpolants can provide the indexed predicates needed to rule out a given MSE, thus guaranteeing refinement progress.

The hope is that, by restricting the analysis to a smaller set of concrete behaviors, the MSE approach may significantly lessen the burden on the first-order prover used for interpolant generation. Of course, it remains to be seen whether in practice the use of abstract counterexamples in IPA is more efficient than the alternatives, such as proof-based abstraction [15] or similar approaches based on weakest preconditions [6].

We also think it is possible that MSE's can be applied to refinement of other non-disjunctive abstractions, such as the partially-disjunctive shape abstractions of [12].

References

1. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207, 1999.
2. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, pages 154–169, 2000.
3. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *POPL*, pages 238–252, 1977.

4. W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symbolic Logic*, 22(3):269–285, 1957.
5. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003.
6. B. S. Gulavani and S. K. Rajamani. Counterexample driven refinement for abstract interpretation. In *TACAS*, pages 474–488, 2006.
7. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL*, pages 232–244, 2004.
8. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In H. Hermanns and J. Palsberg, editors, *TACAS*, volume 3920 of *LNCS*, pages 459–473. Springer, 2006.
9. J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symbolic Logic*, 62(2):457–486, June 1997.
10. R. P. Kurshan. *Computer-Aided-Verification of Coordinating Processes.* Princeton University Press, 1994.
11. S. K. Lahiri and R. E. Bryant. Predicate abstraction with indexed predicates. *ACM Trans. Comput. Log.*, 9(1), 2007.
12. R. Manevich, S. Sagiv, G. Ramalingam, and J. Field. Partially disjunctive heap abstraction. In *SAS*, pages 265–279, 2004.
13. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
14. K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS*, pages 413–427, 2008.
15. K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *TACAS*, pages 2–17, 2003.
16. A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, pages 82–97, 2001.
17. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symbolic Logic*, 62(2):981–998, June 1997.
18. H. Saïdi and S. Graf. Construction of abstract state graphs with PVS. In *CAV*, pages 72–83, 1997.