Thread-Modular Verification with Arbitrary Precision

Alexander Malkis[1], Andreas Podelski[1], and Andrey Rybalchenko[2]

[1] Albert-Ludwigs-Universität Freiburg
[2] EPFL

Abstract. State explosion is the curse of concurrency. Thread-modular verification of multithreaded programs is a promising method that circumvents the state explosion. The method trades its polynomial complexity for a loss of precision that limits the verification power of the method. In this paper we show why this limit can be removed and how. Our work is based on the fact that thread-modular verification is abstract interpretation with Cartesian abstraction. We show that one can parameterize the abstraction and thus adapt the degree of precision to the specific verification problem. The parameterized thread-modular verification is still polynomial in the number of threads. We formalize our approach in the abstract interpretation framework, giving a sufficient condition under which a polynomial-time abstraction can be parameterized without loss of polynomial-time complexity.

1 Introduction

Verification of multithreaded software is known to be a hard problem (see e.g. [9]). The number of states of multithreaded programs grows exponentially with the number of threads. This is known as the state-explosion problem. One way to circumvent the state explosion problem is to design *thread-modular verification* algorithms that reason about concurrent software in modular way. Modularity allows one to avoid the explicit construction of the global state space by considering each thread in isolation (see [5], [7] and [11]).

The thread-modular verification algorithm of Flanagan and Qadeer (see [5]) circumvents state explosion by introducing an important overapproximation. As pointed out by Flanagan and Qadeer, the overapproximation is too coarse for many verification problems, e.g. mutual exclusion protocols. The reason for this is that the algorithm "forgets" some of the dependencies between threads and even "forgets" much of the temporal succession of states in a run. It is possible to view this algorithm as an attribute-independent method, characterizing the overapproximation in the abstract interpretation framework and reformulating the Flanagan-Qadeer algorithm as an abstract interpretation with a form of Cartesian abstraction (cf. [11], [12]).

The question is whether there is a way to modify the algorithm such that its abstraction can be parameterized. This is in analogy to the situation in abstract interpretation (see [2]) where the quality of program analysis depends on the judicious choice of an abstract domain. In this paper we will present such a parameterized method that extends thread-modular verification and is still polynomial in the number of threads.

We show how the approach can be formalized in the framework of abstract interpretation. We show that it is always possible to choose the parameter in such a way that the parameterized abstraction is complete for a concrete verification task. The parameter separates the state space in two parts: the one which is still subjected to the underlying abstraction and the part which is not subjected to the abstraction, but assumed to be reachable by definition. If this second part is actually reachable, then diminishing it leads to less precise analysis. We discover sufficient conditions that allow preserving polynomial-time complexity of the underlying abstraction.

Outline of the paper: First we explain our general approach in the abstract interpretation framework. We say what does it mean to parameterize an abstraction. We describe how different choices of the parameter influence precision. Especially we prove that the parameter can be chosen in such a way that abstract fixpoint checking is complete. We give a sufficient condition so that the parameterization introduces only a polynomial-time increase in complexity.

Then we apply this approach to multithreaded programs. First we define our multithreaded program model. Then we make the reader acquainted with Cartesian abstract interpretation for multithreaded programs. After that we introduce parameterized Cartesian abstraction. We define the abstract domain and abstraction and concretization maps, that allow using the standard forward abstract fixpoint checking algorithm. After that we look at the time and space requirements of the parameterized Cartesian successor computation. Then we present examples and a large class of programs that can be handled by the approach. At last, we describe the related work.

2 Parameterizing Abstractions

2.1 Preliminaries

In the framework of abstract interpretation (cf. [1] for introduction), abstract fixpoint checking is usually described by a concrete domain D, an abstract domain $D^{\#}$, a Galois connections (α, γ) between them and a monotone successor operator F. We describe characterize a class of such tuples $(D, D^{\#}, \alpha, \gamma, F)$ so that parameterized abstractions enjoy arbitrary precision and abstract fixpoint checking takes polynomial time.

Here and from now on, the application of a map f on an argument x is written as fx; the parentheses are used for delimiting subterms in unclear cases. We also write fg for the composition $f \circ g$ with $(f \circ g)x = f(gx)$. A lattice is *boolean* if it is complemented and distributive.

Assumption: Let D be a complete boolean lattice, $D^{\#}$ a complete lattice, (α, γ) a Galois-connection with $\alpha : D \to D^{\#}$ and $\gamma : D^{\#} \to D$. For the bottom element of D write $\emptyset$, write $\cup$ for the join, $\cap$ for the meet, and X^c for the complement of $X \in D$. The bottom element of $D^\#$ is denoted by $\bot$, the join is $\sqcup$, the meet is $\sqcap$. Let the concretization map bottom to bottom, i.e. $\gamma\bot = \emptyset$. Further let $F : D \to D$ be monotone, i.e. $\forall\, x, y \in D : x \subseteq y \Rightarrow Fx \sqsubseteq Fy$.

As usual, we denote the image of the abstraction map by $D^{\#^+} := \alpha D$. The image of $D^{\#^+}$ under the concretization map is $D^+ := \gamma D^{\#^+}$.

2.2 Exceptional Abstraction

The idea of parameterizing an abstraction is introducing a concrete element $E \in D$ (called the Exception Element), which is itself not subjected to abstraction and everything below it also should not be subjected to abstraction. Formally, consider the "exceptional abstraction" and "exceptional concretization" maps

$$\alpha_E : D \to D, \quad X \mapsto X \cap E^c$$

and

$$\gamma_E : D \to D, \quad X \mapsto X \cup E\,.$$

Proposition 1. *The pair* (α_E, γ_E) *is a Galois connection. Formally:*

$$\forall\, X, Y \in D : \quad \alpha_E X \subseteq Y \Leftrightarrow X \subseteq \gamma_E Y$$

Proof. Let $X, Y \in D$.
"$\Rightarrow$": Let $\alpha_E X \subseteq Y$, i.e. $X \cap E^c \subseteq Y$. Then $X \subseteq X \cup E = (X \cup E) \cap$ "top of D" $= (X \cup E) \cap (E^c \cup E) \overset{\text{distributivity}}{=} (X \cap E^c) \cup E \overset{X \cap E^c \subseteq Y}{\subseteq} Y \cup E = \gamma_E Y$.
"$\Leftarrow$": Let $X \subseteq \gamma_E Y$, i.e. $X \subseteq Y \cup E$. Then $\alpha_E X = X \cap E^c \overset{X \subseteq Y \cup E}{\subseteq} (Y \cup E) \cap E^c \overset{\text{distributivity}}{=} (Y \cap E^c) \cup \underbrace{(E \cap E^c)}_{\bot} = Y \cap E^c \subseteq Y$. $\square$

The composition $(\alpha\alpha_E, \gamma_E\gamma)$ of Galois connections is itself a Galois-connection. We call it the *parameterized* Galois connection. Let init $\in D$ be any element. Our analysis computes lfp $\lambda Y.\, \alpha\alpha_E(\text{init} \cup F\gamma_E\gamma Y)$. The concretization of this least fixpoint $\gamma_E\gamma(\text{lfp } \lambda Y.\, \alpha\alpha_E(\text{init} \cup F\gamma_E\gamma Y))$ overapproximates the least fixpoint of init $\cup F$ by a standard result of the abstract interpretation framework (cf. [1], Prop. 13), so soundness of the analysis is guaranteed.

2.3 Influence of Exceptional Element on Precision

Now we discuss in detail how the choice of the Exception Element influences precision of the abstract fixpoint when using the parameterized Galois connection.

If $E = \emptyset$, then the exceptional abstraction and concretization maps are identities. That means no difference in comparison to the abstract fixpoint computation with (α, γ).

Since for a monotone map F the map $\lambda x.\, \mathrm{init} \cup Fx$ is also monotone, the properties derived for an arbitrary monotone F also hold for $\lambda x.\, \mathrm{init} \cup Fx$. In this section we formulate everything for F.

Before proceeding, first remember that a postfix point of a map $F : D \to D$ is any x with $Fx \subseteq x$. Second, according to the Knaster-Tarski fixpoint theorem, the least fixpoint of a monotone F exists and is the meet of its all postfix points (cf. [16], Thm. 1), i.e. $\mathrm{lfp}\, F = \bigcap\{x \mid Fx \subseteq x\}$.

Proposition 2. *An element is a postfix point of a monotone map if and only if the parameterized concretization of the abstract fixpoint (with Galois connection parameterized by this element) gives exactly this element. Formally:*

$$\forall\, F \in (D \xrightarrow{mon} D),\ E \in D :\ \ FE \subseteq E \ \Leftrightarrow\ \gamma_E\gamma(\mathrm{lfp}\,(\alpha\alpha_E F\gamma_E\gamma)) = E\,.$$

Proof. Let F be monotonic and E some element.
"$\Rightarrow$": Let $FE \subseteq E$. Since $\gamma\bot = \emptyset$ by the assumption on the concretization map, we have $\gamma_E\gamma\bot = E$ and by postfix point property of E we have $F\gamma_E\gamma\bot \subseteq E$, and thus $\alpha\alpha_E F\gamma_E\gamma\bot = \bot$ (that alpha maps bottom to bottom follows from the definition of a Galois connection). So the abstract fixpoint $\mathrm{lfp}\,(\alpha\alpha_E F\gamma_E\gamma) = \bot$ is trivial, thus its concretization is $\gamma_E\gamma\mathrm{lfp}\,(\alpha\alpha_E F\gamma_E\gamma) = E$.
"$\Leftarrow$": This holds in general for all Galois connections and not only for $(\alpha\alpha_E, \gamma_E\gamma)$. So let $(\bar\alpha, \bar\gamma)$ be a Galois connection between the concrete domain D and the abstract domain $D^\#$ and $E = \bar\gamma\mathrm{lfp}\,(\bar\alpha F\bar\gamma)$. Then $FE = F\bar\gamma\mathrm{lfp}\,(\bar\alpha F\bar\gamma)$. The map $\bar\gamma\bar\alpha$ is extensive (a map ρ is called *extensive* iff $x \subseteq \rho x$ for all x from the domain of ρ), so $FE \subseteq (\bar\gamma\bar\alpha)F\bar\gamma\mathrm{lfp}\,(\bar\alpha F\bar\gamma) = \bar\gamma(\bar\alpha F\bar\gamma)\mathrm{lfp}\,(\bar\alpha F\bar\gamma) = \bar\gamma\mathrm{lfp}\,(\bar\alpha F\bar\gamma) = E$. $\square$

For program analysis this means: whenever we neither gain nor lose any precision by taking a set as an exception element, this set is an inductive program invariant and vice versa.

Since the least fixpoint of F exists and is a postfix point, we obtain

Corollary 3. *There is an exception set so that abstract fixpoint computation with the Galois connection parameterized by this set gives the exact answer. Formally:*

$$\exists\, E :\ \gamma_E\gamma(\mathrm{lfp}\,(\alpha\alpha_E F\gamma_E\gamma)) = \mathrm{lfp}\, F\,.$$

For program analysis this means that the parameterized analysis is complete in the sense that there is a choice of the exception set that leads to the best inductive invariant.

Corollary 4. *An element A is an abstract postfix point if and only if the abstract fixpoint computation with Galois connection parameterized by the concretization of A returns the concretization of A. Formally:*

$$\forall\, A \in D^\# :\quad \alpha F\gamma A \sqsubseteq A \ \Leftrightarrow\ \gamma_{\gamma A}\gamma\mathrm{lfp}\,(\alpha\alpha_{\gamma A}F\gamma_{\gamma A}\gamma) = \gamma A\,.$$

Proof. Since (α, γ) is a Galois connection, $\alpha F\gamma A \sqsubseteq A \Leftrightarrow F\gamma A \subseteq \gamma A$. The result follows with $E = \gamma A$ from Prop. 2. $\square$

For program analysis this means: whenever A is an abstract inductive invariant, the abstract analysis with parameter γA neither increases nor decreases the precision compared to γA and vice versa. Especially one cannot increase precision by taking the least abstract fixpoint $\operatorname{lfp}(\alpha F \gamma)$ as an exception set.

Now we show how the fixpoints are related when the parameter increases or decreases.

Proposition 5. *Let A be any element and X be a result of abstract fixpoint computation with abstraction parameterized by A. Then taking as an exception set any element between A and the parameterized (by A) concretization of X and performing abstract fixpoint checking parameterized with this new element increases precision both in the abstract and in the concrete. Formally:*
Let $A, B \in D$, $F : D \xrightarrow{mon} D$, $X = \operatorname{lfp}(\alpha\alpha_A F \gamma_A \gamma)$, $A \subseteq B \subseteq \gamma_A \gamma X$ and $Y = \operatorname{lfp}(\alpha\alpha_B F \gamma_B \gamma)$. Then
1. *$Y \sqsubseteq X$.*
2. *For $U = \gamma_A \gamma X$ and $V = \gamma_B \gamma Y$ holds $V \subseteq U$.*

Proof.
1. From $B \subseteq A \cup \gamma X$ follows $B \cup \gamma X \subseteq A \cup \gamma X$. From $A \subseteq B$ follows $B^c \subseteq A^c$. We have $\alpha\alpha_B F \gamma_B \gamma X = \alpha((F(B \cup \gamma X)) \cap B^c) \sqsubseteq$ [monotonicity of F and α] $\alpha((F(A \cup \gamma X)) \cap A^c) = \alpha\alpha_A F \gamma_A \gamma X = X$. So X is a postfix point of $\alpha\alpha_B F \gamma_B \gamma$, while Y is its least fixpoint, which is by Knaster-Tarski less than or equal to any of its postfix points. Thus $Y \sqsubseteq X$.
2. From monotonicity of the concretization map follows $\gamma Y \subseteq \gamma X$ and thus $B \cup \gamma X \subseteq A \cup \gamma X$ implies $V = B \cup \gamma Y \subseteq A \cup \gamma X = U$. □

For program analysis this means that under certain circumstances choosing a greater exception set leads to a smaller invariant, even if the exception sets are not contained in the most precise program invariant. This could be helpful, for instance, if a larger exception set can be represented more efficiently. However, if both A and B are contained in the least invariant, then the precision of abstract fixpoint checking is increased for sure:

Corollary 6. *Let A, B be any two comparable elements below the least concrete fixpoint. Then parameterizing the abstract fixpoint computation by the greater element leads to a more precise abstract fixpoint than parameterizing it by the smaller element. Formally:*
Let $A \subseteq B \subseteq \operatorname{lfp} F$, $X = \operatorname{lfp}(\alpha\alpha_A F \gamma_A \gamma)$, $Y = \operatorname{lfp}(\alpha\alpha_B F \gamma_B \gamma)$, $U = \gamma_A \gamma X$, and $V = \gamma_B \gamma Y$. Then $Y \sqsubseteq X$ and $V \subseteq U$.

Proof. The standard result of the abstract interpretation framework (cf. [1], Prop. 13) applied to $(\alpha\alpha_A, \gamma_A \gamma)$ implies $\operatorname{lfp} F \subseteq \gamma_A \gamma X$. From $B \subseteq \operatorname{lfp} F$ follows $B \subseteq \gamma_A \gamma X$. The result follows from Prop. 5. □

For program analysis this means that as long as the exception set grows, but remains below the best invariant, the precision of the analysis grows too.

Proposition 7. *Assume some postfix point of F is an exceptional concretization of the concretization of an abstract element. Then computation of the abstract fixpoint parameterized by the exception element gives an element whose parameterized concretization is less than or equal to that postfixed point of F.*

Formally: Let $E \in D$, $Y \in D^{\#}$, $X = \gamma Y$, $R = \gamma_E X$, $F : D \xrightarrow{mon} D$, $FR \subseteq R$. Then $\gamma_E \gamma(\text{lfp}\,(\alpha\alpha_E F \gamma_E \gamma)) \subseteq R$.

Proof. Remember that for any Galois connection $(\bar{\alpha}, \bar{\gamma})$ between D and $D^{\#}$ holds that $\bar{\alpha}\bar{\gamma}$ is reductive (i.e. $\forall\, y \in D^{\#} : \bar{\alpha}\bar{\gamma}y \sqsubseteq y$). Moreover both abstraction and concretization are monotonic. Then $\alpha_E R = R \cap E^c = (E \cup X) \cap E^c = (E \cap E^c) \cup (X \cap E^c) = \emptyset \cup (X \cap E^c) \subseteq X = \gamma Y$. So $\gamma_E \gamma \alpha \alpha_E R \subseteq \gamma_E \gamma \alpha \gamma Y \subseteq$ [$\alpha\gamma$ reductive] $\gamma_E \gamma Y = \gamma_E X = R$. Then $\alpha\alpha_E F \gamma_E \gamma(\alpha\alpha_E R) = \alpha\alpha_E F(\gamma_E \gamma \alpha \alpha_E) R \sqsubseteq \alpha\alpha_E F R \sqsubseteq$ [since $FR \subseteq R$] $\alpha\alpha_E R$. So $\alpha\alpha_E R$ is a postfix point of $\alpha\alpha_E F \gamma_E \gamma$ and thus greater than or equal to its least fixpoint. So $\gamma_E \gamma(\text{lfp}\,(\alpha\alpha_E F \gamma_E \gamma)) \subseteq \gamma_E \gamma \alpha \alpha_E R \subseteq R$. □

2.4 Polynomial Exceptional Abstraction

In order to be able to argue about polynomial run time, we measure a domain by the number of elements in its longest chain and by the maximal length of the binary representation of its elements. We take the maximum of both values as the *measure* of the domain and formulate the run time as a function of this maximum.

Assumption:

- The map F is a join-morphism.
- The abstract domain is finite. Transforming the elements of the abstract domain to and from binary string takes polynomial time. Formally: if m is the measure of $D^{\#}$ then there is a pair of maps

$$\phi : D^{\#} \hookrightarrow \{0,1\}^m \,,$$
$$\phi^{-1} : \{0,1\}^m \to D^{\#} \,,$$

  so that ϕ is one-to-one, ϕ^{-1} is a left inverse of ϕ, ϕ is polynomial-time computable and ϕ^{-1} is polynomial-time computable on the image of ϕ.
- The abstract join operation needs polynomial time in m.

In the following, 2^X is the power set of X. The next assumption on F is (in contradiction to the assumptions before) a strong restriction, so we call it a

Hypothesis: The function F maps the concretization of every abstract element to a concrete join of concretizations of polynomial many abstract elements. Formally speaking, there exists a polynomial-time computable map

$$\bar{F} : {D^{\#}}^{+} \to 2^{{D^{\#}}^{+}} \,,$$
$$y \mapsto Y \,,$$

where $|Y| \le m^k$ (for a constant $k \in \mathbb{N}$ independent on m) and where

$$F\gamma y = \bigcup \gamma Y \,.$$

Proposition 8. *Under the assumptions and the Hypothesis, there is a RAM program and for any exception element E there is a data structure for this program so that the program computes the best correct approximation $F_E^{\#} = \alpha\alpha_E F\gamma_E\gamma$ on ${D^{\#}}^+$ in polynomial time in the measure of the abstract domain.*

Proof. Let $y \in {D^{\#}}^+$. We have

$$\begin{aligned} F_E^{\#} y = \alpha(E^c \cap F(E \cup \gamma y)) &= [F \text{ join-morphism}] \\ = \alpha(E^c \cap ((FE) \cup (F\gamma y))) &= [\text{distributivity}] \\ = \alpha((E^c \cap FE) \cup (E^c \cap F\gamma y)) &= [\alpha \text{ join-morphism}] \\ = \alpha(E^c \cap FE) \sqcup \alpha(E^c \cap F\gamma y). \end{aligned}$$

Since E is a constant independent on y, the expression $\alpha(E^c \cap FE) =: E'$ is also constant and can be computed once; we view E' as a part of representation of E. The join operation needs polynomial time, so is suffices to show that the map $y \mapsto \alpha(E^c \cap F\gamma y)$ is polynomial computable. The hypothesis implies that this expression equals

$$\alpha(E^c \cap \bigcup \gamma \bar{F} y) = [\text{distributivity}]\ \alpha(\bigcup \{E^c \cap \gamma y' \mid y' \in \bar{F} y\}) =$$
$$= [\alpha \text{ join-morphism}]\ \bigsqcup \{\alpha((\gamma y') \cap E^c) \mid y' \in \bar{F} y\}\,.$$

Since $\bar{F}y$ has polynomial many elements and the abstract join $\sqcup$ is polynomial-time computable, it suffices to show that the map $y' \mapsto \alpha((\gamma y') \cap E^c)$ is computable in polynomial time. Notice that $\alpha((\gamma y') \cap E^c) = \phi^{-1}\phi\alpha((\gamma\phi^{-1}\phi y') \cap E^c)$. Since both ϕ and ϕ^{-1} are polynomial-time computable, if suffices to show that the map

$$\{0,1\}^m \to \{0,1\}^m\,, v \mapsto \phi\alpha((\gamma\phi^{-1}v) \cap E^c)$$

is polynomial-time computable in m at least on the image of ϕ. This map, as any map from $\{0,1\}^m \to \{0,1\}^m$, can be represented by a decision tree of size $m2^m$ and depth m (or even a multiterminal binary decision diagram) on a RAM with logarithmic cost measure and built-in addition (cf. [13]). An evaluation takes $O(m^2)$ time. This map doesn't depend on F, so we view the decision tree as the second part of the representation of E. ◻

Theorem 9. *Under the assumptions and the Hypothesis, there is a representation of the exception element so that abstract fixpoint computation takes polynomial time in the measure of the abstract domain. Formally:*

Computing

$$\operatorname{lfp}\ \lambda Y.\ \alpha\alpha_E(\operatorname{init} \cup F\gamma_E\gamma Y)$$

takes polynomial time in m.

Proof. The usual abstract fixpoint iteration sequence is $X^0 = \bot$ and $X^{i+1} = \alpha\alpha_E(\operatorname{init} \cup F\gamma_E\gamma X^i)$. This sequence is monotone and stabilizes in a finite number of steps. The abstraction maps α and α_E are join-morphisms, so $X^{i+1} =$

$\alpha\alpha_E\text{init} \sqcup \alpha\alpha_E F\gamma_E\gamma X^i$. The first term is constant, the second can be computed in polynomial time as just shown, the join operation is polynomial. So the next term of the sequence $(X^i)_i$ is computable in polynomial time from the previous one. Since the maximal chain length is at most m, the number of iterations till the sequence stabilizes is at most m. $\square$

3 Thread-Modular Verification

3.1 Multithreaded Programs

A *multithreaded program* is a tuple

$$(\text{Glob}, \text{Loc}, (\rightarrow_i)_{i=1}^n, \text{init}),$$

where Glob and Loc are any sets, each $\rightarrow_i$ is a subset of $(\text{Glob} \times \text{Loc})^2$ (for $1 \leq i \leq n$) and $\text{init} \subseteq \text{Glob} \times \text{Loc}^n$.
The meaning of different components of the multithreaded program is the following:

- Loc contains valuations of local variables (including the program counter) of any thread, we call it the *local store* of the thread (without loss of generality we assume that all threads have equal local stores);
- Glob contains valuations of shared variables, we call it the *global store*;
- the elements of $\text{States} = \text{Glob} \times \text{Loc}^n$ are called *program states*, the elements of $Q = \text{Glob} \times \text{Loc}$ are called *thread states*, the projection on the global store and the ith local store is the map

$$\pi_{\{0,i\}} : 2^{\text{States}} \rightarrow 2^Q, \quad S \mapsto \{(g, l_i) \mid (g, l) \in S\}\,;$$

- the relation $\rightarrow_i$ is a transition relation of the ith thread ($1 \leq i \leq n$);
- init is a set of initial states.

The program is equipped with the usual interleaving semantics. This means that if a thread makes a step, then it may change its own local variables and the global variables but may not change the local variables of another thread; a step of the whole program is a step of some of the threads. The successor operation maps a set of program states to the set of their successors:

$$\begin{aligned} &\text{post} : 2^{\text{States}} \rightarrow 2^{\text{States}} \\ &S \mapsto \{(g', l') \in \text{States} \mid \exists\, (g, l) \in S,\, i \in \{1, ..., n\} : (g, l_i) \rightarrow_i (g', l'_i) \\ &\qquad\qquad \text{and } \forall\, j \neq i : l_j = l'_j\}. \end{aligned}$$

We are interested in proving safety properties of multithreaded programs. Each safety property can be encoded as a reachability property and each reachability property can be encoded as reachability between a pair of states. So we are interested whether there is a computation of any length $k \geq 0$ that starts in an initial state and ends in a single user-given error state f, formally:

$$\exists\, k \geq 0 : \ f \in \text{post}^k(\text{init})\,.$$

The state explosion problem in context of multithreaded programs refers to the fact that the number of program states is exponentially large in the number of threads n. We don't address the problem of growing state space due to the number of variables, which is also common to sequential programs.

3.2 Cartesian abstract interpretation

In previous work (see [11]) the authors presented the theory of Cartesian abstraction for multithreaded programs. Here we repeat a necessary part of this background for clarity.

We present a concrete and an abstract domain and a Galois connection between them that allows us to do abstract fixpoint checking. The definitions below extend the usual notion of the dependence-free abstraction (as, for instance, formulated by Cousot in [3]):

$D = 2^{\text{States}}$ is the set underlying the concrete lattice,
$D^{\#} = (2^{\text{Glob}\times\text{Loc}})^n$ is the set underlying the abstract lattice,
$\alpha_{\text{cart}} : D \to D^{\#},$
$\quad S \mapsto \left(\pi_{\{0,i\}} S\right)_{i=1}^{n}$
is the abstraction map, which projects a set of program states to the tuple of sets of thread states, so that the ith component of a tuple contains all states of the ith thread that occur in the set of program states.
$\gamma_{\text{cart}} : D^{\#} \to D,$
$\quad T \mapsto \{(g,l) \mid \forall\, i \in \{1,...,n\} : (g,l_i) \in T_i\}$
is the concretization map that combines a tuple of sets of thread states to a set of program states by putting only those thread states together that have equal global part.

The ordering on the concrete domain D is inclusion, the least upper bound is the union $\cup$, the greatest lower bound is the intersection $\cap$, the complement X^c of a set X.

The ordering on the abstract domain $D^{\#}$ is the product ordering, i.e. $T \sqsubseteq T'$ if and only if $T_i \subseteq T'_i$ for all $i \in \{1,...,n\}$. The least upper bound $\sqcup$ is componentwise union, the greatest lower bound $\sqcap$ is componentwise intersection. So the abstract lattice is complete. The bottom element is the tuple of empty sets $\perp = (\emptyset)_{i=1}^{n}$.

Notice that the image of the concrete lattice under abstraction is

$$D^{\#^+} = \{(R_i)_{i=1}^{n} \mid \forall g \in \text{Glob} :\ (\exists i \in \mathbb{N}_n : (g,\_) \in R_i) \Rightarrow \forall j \in \mathbb{N}_n : (g,\_) \in R_j\}\,,$$

where underscore (_) means "anything", i.e. an innermost existentially quantified variable, and $\mathbb{N}_n = \{1,...,n\}$ is the set of first n natural numbers.

The following is an extension of the corresponding standard result (cf. [3]).

Proposition 10. *The pair of maps $(\alpha_{\text{cart}}, \gamma_{\text{cart}})$ is a Galois connection, i.e. for all $S \in D, T \in D^{\#}$ holds*

$$\alpha_{\text{cart}} S \sqsubseteq T \textit{ iff } S \subseteq \gamma_{\text{cart}} T\,.$$

Proof. "⇒": Let $(g, l) \in S$. Let $T' = \alpha_{\text{cart}} S$. Then by definition of α_{cart} we have for all $i \in \{1, ..., n\}$ that $(g, l_i) \in T'_i \subseteq T_i$. So $(g, l) \in \gamma_{\text{cart}} T$ by definition of γ_{cart}.
"⇐": Let $T' = \alpha_{\text{cart}} S$. Fix some $i \in \{1, ..., n\}$ and let $(g, l_i) \in T'_i$. By definition of α_{cart} for each other $j \neq i$ there is an l_j so that the tuple containing all these elements $(g, (l_j)_{j=1}^n)$ is in S. But $S \subseteq \gamma_{\text{cart}} T$, so $(g, (l_j)_{j=1}^n) \in \gamma_{\text{cart}} T$. By definition of γ_{cart} we have $(g, l_i) \in T_i$. So $T'_i \subseteq T_i$. Since i was arbitrarily chosen, we have $T' \sqsubseteq T$ by definition of $\sqsubseteq$. □

4 Parameterized Cartesian Abstraction

First we prove why the conditions for parameterized abstraction to work are satisfied. We show what run time and what space requirements are needed. After that we give another algorithm computing the abstract post with different time-space requirements, thus discovering a time-space tradeoff.

4.1 Polynomial Time

Now we show why the conditions of the Theorem 9 are met for the Cartesian abstraction/concretization maps on the above domains D and $D^{\#}$.

First, the concrete lattice D is boolean and complete. Then the concretization map γ_{cart} maps $\bot = (\emptyset)_{i=1}^n$ to $\emptyset$. The successor map post distributes over union, so it is a join-morphism. The maximal chain length of $D^{\#}$ is $n|\text{Glob}||\text{Loc}| + 1$. A single abstract element is an n-tuple $(R_1, ..., R_n)$ of subsets R_i of Glob × Loc, which can be coded in polynomial time by $n|\text{Glob}||\text{Loc}|$ boolean variables corresponding to the answers to the questions "$(g, l) \in R_i$" for $g \in \text{Glob}$, $l \in \text{Loc}$ and $i \in \{1, ..., n\}$. The abstract join operation $(R_i)_{i=1}^n \sqcup (R'_i)_{i=1}^n = (R_i \cup R'_i)_{i=1}^n$ takes time proportional to n and the maximal possible sizes of the sets R_i, R'_i $(1 \leq i \leq n)$.

To prove the hypothesis, assume we have an abstract element $(R_i)_{i=1}^n$, which can also be written as $\bigsqcup_{g \in \text{Glob}} (\{g\} \times R_i^g)_{i=1}^n$ where $R_i^g = \{l \mid (g, l) \in R_i\}$ for $1 \leq i \leq n$. Its concretization is $\bigcup_{g \in \text{Glob}} (\{g\} \times \prod_{i=1}^n R_i^g)$. So in order to show that the successors of this set are a union of polynomially many elements of D^+, it suffices to show that for any $g \in \text{Glob}$ the successor set of $\{g\} \times \prod_{i=1}^n R_i^g$ is a union of polynomial many elements of D^+. For each $g \in \text{Glob}$ we have $\text{post}(\{g\} \times \prod_{i=1}^n R_i^g) =$

$$\bigcup_{(g,l_i) \to_i (g', l'_i)} \{g'\} \times R_1^g \times ... \times R_{i-1}^g \times \begin{Bmatrix} \{l'\} \text{ , if } l \in R_i^g \\ \emptyset \quad \text{ , otherwise} \end{Bmatrix} \times R_{i+1}^g \times ... \times R_n^g \,,$$

the number of Cartesian products being bounded by the number of transitions of all threads, which is at most $n(|\text{Loc}||\text{Glob}|)^2$. So we have proven that parameterized Cartesian abstraction is polynomial.

An exact analysis gives the run time $O(G^2 L^4 n^3 \log(LGn))$ for a single computation of the abstract post operator, while the data structure for the exception set needs space $O(GL^2n^22^{Ln})$ where $L = |\mathrm{Loc}|$ and $G = |\mathrm{Glob}|$ and the machine model is RAM with logarithmic cost measure. Using unit cost measure instead one can get $O(G^2L^3n^2)$ run time for a single iteration. So the whole analysis, i.e. iteration till fixpoint, takes cubic time in the number of threads under the unit cost measure. The proofs of the run times are left to the reader as an exercise.

4.2 Polynomial space

Now we try to get rid of the exponential space in the size of the local store. We develop an algorithm for computing abstract successor operator with polynomial time in the number of threads and in the size of the exception set. Since two different exception sets lead in general to different abstract fixpoints, a representation of E needs at least $|\mathrm{Glob}||\mathrm{Loc}|^n$ space in a worst case.

Let

$$\mathrm{post}_{E,\mathrm{cart}} = \alpha_{\mathrm{cart}} \circ \alpha_E \circ \mathrm{post} \circ \gamma_E \circ \gamma_{\mathrm{cart}}$$

be the best correct approximation of post.

For the following technical lemma, the numbers n and L are any natural numbers and are not necessarily related to the multithreaded program.

Lemma 11. *There is an algorithm, that for n finite sets A_i $(1 \leq i \leq n)$ with at most L elements each (represented so that $|A_i|$ for each i is computable in polynomial time in L and in n), computes the size of their product $|\prod_{i=1}^n A_i|$ in polynomial time in L and in n.*

Proof. The number is $\prod_{i=1}^n |A_i|$. Each number $|A_i|$ has $O(\log L)$ digits, and the number $|A_1| \cdots |A_{i-1}|$ has $O((i-1)\log L)$ digits. So multiplying these two numbers needs $O((i-1)\log^2 L)$ primitive additions. Summing over i gives $O(n^2 \log^2 L)$ primitive additions. ☐

Proposition 12. *The parameterized Cartesian successor operator $\mathrm{post}^{\#}_{E,\mathrm{cart}}$ is computable in polynomial time in n, $|\mathrm{Glob}|$, $|\mathrm{Loc}|$ and $|E|$, the representation of E taking space $O(|E|)$.*

Proof. Let E be represented as a list of tuples, sorted according to the global component. Following the lines of the proof of Proposition 8, we consider the innermost computation step, namely the computation of the map

$$f_E : D^{\#} \to D^{\#}, y' \mapsto \alpha_{\mathrm{cart}}((\gamma_{\mathrm{cart}} y') \setminus E)\,.$$

So let y' be represented by the sets $R_i^g = (\{g\} \times \mathrm{Loc}) \cap R_i$ for $g \in \mathrm{Glob}$, $1 \leq i \leq n$ as $y' = \sqcup_{g \in \mathrm{Glob}} (R_i^g)_{i=1}^n$. Then

$$f_E y' = \bigsqcup_{g \in \mathrm{Glob}} f_E((R_i^g)_{i=1}^n)\,.$$

Let E be represented by sets $E^g = (\{g\} \times \mathrm{Loc}^n) \cap E$. We get all E^g from E by going through the list E once. Then it suffices to compute $F_E((R_i^g)_{i=1}^n) =$

$\alpha_{\text{cart}}((\gamma_{\text{cart}}(R_i^g)_{i=1}^n) \setminus E) = \alpha_{\text{cart}}((\gamma_{\text{cart}}(R_i^g)_{i=1}^n) \setminus E^g) = F_{E^g}((R_i^g)_{i=1}^n$ once for each g and join the results. So let's fix g. Since in computation of the last term the global part remains constant g, we may as well forget it and use the standard Cartesian abstraction/concretization maps

$$\alpha_c : 2^{(\text{Loc}^n)} \to (2^{\text{Loc}})^n, S \mapsto (\pi_i S)_{i=1}^n\,,$$
$$\gamma_c : (2^{\text{Loc}})^n \to 2^{(\text{Loc}^n)}, T \mapsto \prod_{i=1}^n T_i\,,$$

computing $\alpha_c((\prod_{i=1}^n A_i) \setminus B)$ for $A_i = \{l_i \mid (g, l_i) \in R_i\}$ $(1 \le i \le n)$ and $B = \{l \mid (g, l) \in E\}$. By going through the list B once we construct $B' = B \cap \prod_{i=1}^n A_i$. It suffices to compute for each $j \in \mathbb{N}_n$ the projection

$$\pi_j(\alpha_c((\prod_{i=1}^n A_i) \setminus B')).$$

Let us fix some j. We have to determine for $r \in A_j$, whether $r \in \pi_j(\alpha_c((\prod_{i=1}^n A_i) \setminus B'))$. It is the case if and only if there are strictly more tuples in the cross product $\prod_{i=1}^n A_i$ containing r in the jth component than in B'. The number of tuples in $\prod_{i=1}^n A_i$ with r at the jth place is $\prod_{1 \le i \le n, i \ne j} |A_i|$, which can be computed in polynomial time by Lemma 11. The number of tuples in B' where r occurs in the jth component can be determined by a linear scan of B'. $\square$

By definition, the parameterized Cartesian successor operator $\text{post}^{\#}_{E,\text{cart}}(R)$ is monotone in the argument R, so the usual fixpoint iteration of

$$\mathfrak{F} = \lambda R.\ \alpha_{\text{cart}}\alpha_E \text{init} \sqcup \text{post}^{\#}_{E,\text{cart}} R\,,$$

produces elements on a chain:

$$\bot \sqsubseteq \mathfrak{F}\bot \sqsubseteq \mathfrak{F}\mathfrak{F}\bot \sqsubseteq \mathfrak{F}\mathfrak{F}\mathfrak{F}\bot \sqsubseteq \ldots$$

The longest chains in the abstract domain $D^{\#} = (2^Q)^n$ have $1 + n|Q| \le 1 + n|\text{Glob} \times \text{Loc}| = O(nGL)$ elements where G is the size of the global store and L the size of a local store. Thus the parameterized Cartesian post operator enjoys the following property.

Theorem 13. *Given a multithreaded program and an exception set E stored as a list, abstract fixpoint checking with the parameterized Cartesian post operator $\text{post}^{\#}_{E,\text{cart}}$ takes polynomial time in the size of program description and in the size of the exception set.*

Formally: There is an algorithm that for each $E \subseteq States$ stored as a list computes

$$\text{lfp}\ \ \lambda R.\, \alpha_{\text{cart}}\alpha_E \text{init} \sqcup \text{post}^{\#}_{E,\text{cart}} R$$

in polynomial time in n, $|\text{Glob}|$, $|\text{Loc}|$, $|\text{init}|$ and $|E|$.

4.3 Exception Set as Union of Maximal Cartesian Products

There is a data structure for the exception set that is simple, needs polynomial space for an important large class of real-world programs and allows polynomial time abstract fixpoint computation in the size of this data structure. First we need a technical result from the set theory.

Assumption:

Let D be any complete lattice with order $\leq$ (strict order $x < y$ iff ($x \leq y$ and $x \neq y$)), and assume we have fixed some "generating" subset of D so that each element of D can be written as a join of some elements of the generating subset. Further let $Y \subseteq D$ be any set that contains the generating set so that the supremum of each chain in Y belongs to Y.

For $a \in D$, an element $y \in D$ is called *a-maximal*, if it is in Y, is less than or equal to a and there is no other greater element of Y that is less than or equal to a, formally: $Y \ni y \leq a$ and $\nexists\, y' \in Y : y < y' \leq a$. A set M is called *maximized*, if $M \subseteq Y$ and $\forall\, y \in Y : y \leq \bigvee M \Rightarrow \exists\, y' \in M : y \leq y'$.

In the following, statements marked by (AC) have a proof that depends on the axiom of choice in case the lattice is infinite, but not if the lattice is finite. It's unknown whether there is a proof independent on AC for infinite lattices.

Proposition 14 (AC). *Let $a \in D$. Then any element of Y less than or equal to a is less than or equal to some a-maximal element.*

Proof. Remember the maximal chain principle (cf. [14]), which says that in each partial order on a set there is a maximal chain. Now let $b \in Y$ and $b \leq a$, then the set $N = \{y \in Y \mid b \leq y \leq a\}$ is partially ordered by $\leq$, so according to the maximal chain principle there is a chain $C \subseteq N$ which cannot be extended. It is also a chain in Y, so $c := \bigvee C \in Y$ by assumption on Y. At the same time $\forall\, y \in C : y \leq a$, so $c \leq a$. If there were another $c' \in Y$ with $c < c' \leq a$, then $c' \in N$ and C could be extended by c' in contradiction to the maximality of C. So c is an a-maximal element greater than or equal to b. $\square$

Proposition 15 (AC). *Each element a of the lattice can be represented as a join of a unique maximized antichain. This maximized antichain contains exactly the a-maximal elements.*

Proof. Let $a \in D$.

First we show existence. So let M contain exactly the a-maximal elements. Two a-maximal elements are incomparable by definition, so M is an antichain.

Since M contains only elements below a, we have $\bigvee M \leq a$. So if $y \in Y$ with $y \leq \bigvee M$ is given, then $y \leq a$, so Prop. 14 gives us some a-maximal element that is greater than or equal to y. So M is maximized.

Now a can be written as $\bigvee A$ for a set of generating elements $A \subseteq Y$ which are all less than or equal to a. Each element in Y which is less than or equal to a is by Prop. 14 less than or equal to some element of M, so $a = \bigvee A \leq \bigvee M$. Together with $\bigvee M \leq a$ we get $a = \bigvee M$.

Now we show uniqueness. It suffices to show that for each two maximized antichains M, M' with $a = \bigvee M = \bigvee M'$ we have $M \subseteq M'$. So let such M and

M' be given and $b \in M$. Then $b \leq a$ and, since M' is maximized and $b \leq \bigvee M'$, there is a $b' \in M'$ with $b \leq b'$. Then $b' \leq a$ and, since M is maximized and $b' \leq \bigvee M$, there is a $b'' \in M$ with $b' \leq b''$. From $b \leq b' \leq b''$ and $b, b'' \in M$ and the antichain property of M follows $b = b' = b''$. This proves $b \in M'$. $\square$

Proposition 16 (AC). *Each maximized set contains the unique maximized antichain with the same join. Formally:*

$$\forall \text{ maximized } A \subseteq Y \ \ \exists^1 M \subseteq A : M \text{ is a maximized antichain and} \bigvee M = \bigvee A\,.$$

Proof. Let $A \subseteq Y$ be maximized and $a = \bigvee A$. According to Prop. 15 there is a maximized antichain M with join a which contains exactly the a-maximal elements. Moreover, it is the only maximized antichain with join a. To show $M \subseteq A$, assume for the purpose of contradiction that there is some a-maximal element y which is not in A. Then $Y \ni y \leq a = \bigvee A$, so by definition of maximized there is $y' \in A$ with $y \leq y'$. From $y \notin A$ follows $y < y'$. From $\bigvee A = a$ follows $y' \leq a$. So there is an element of Y (namely y') greater than y and less than or equal to a, in contradiction to y being a-maximal. $\square$

Now let's consider the Cartesian products. Recall that a function is a set of pairs so that for each first component there is exactly one second component. For an index set I, a *Cartesian product* of sets A_i $(i \in I)$ is the set of maps $\prod_{i\in I} A_i := \{f : I \to \cup_{i\in I} A_i \mid \forall\, i \in I : f(i) \in A_i\}$. For a subset of indices $J \subseteq I$ the *projection* of a subset $A \subseteq \prod_{i\in I} A_i$ on the components J is $\pi_J A = \{f : J \to \cup_{j\in J} A_j \mid \exists\, g \in A : f \subseteq g\}$. A projection on a single index $i \in I$ is $\pi_i A = \{a \in A_i \mid \exists\, g \in A : (i, a) \in g\}$. For a natural number n, the set $A^n := \prod_{i=1}^n A$ is the nth power of A.

Lemma 17 (AC). *Let A_i, B_i be sets indexed by $i \in I$. Then*

$$\prod_{i\in I} A_i \subseteq \prod_{i\in I} B_i \quad \Leftrightarrow \quad ((\forall\, i \in I : A_i \subseteq B_i) \ \ or \ \ \exists\, i \in I : A_i = \emptyset)\,.$$

Proof.

"$\Rightarrow$": Let the Cartesian product of A_i $(i \in I)$ be included in the Cartesian product of B_i $(i \in I)$. If there is an index i so that $A_i = \emptyset$, then the claim holds trivially. Otherwise all A_i are nonempty $(i \in I)$. Now fix some $i \in I$ and let $a \in A_i$. The axiom of choice applied to the sets A_j $(j \in J := I \setminus \{i\})$ gives a choice function $f : J \to \bigcup_{j\in J} A_j$ with $f(j) \in A_j$ $(j \in J)$. For $g := f \cup \{(i, a)\}$ follows that $g \in \prod_{j\in I} A_j$, so $g \in \prod_{j\in I} B_j$, thus $g(i) \in B_i$, so $a \in B_i$. This proves $A_i \subseteq B_i$.

"$\Leftarrow$": If for some $i \in I$ holds $A_i = \emptyset$, then there is no map f with $f(i) \in A_i$, so $\prod_{i\in I} A_i = \emptyset$. Otherwise all A_i are nonempty $(i \in I)$. Now let $f \in \prod_{i\in I} A_i$. Then $\forall\, i \in I : f(i) \in A_i \subseteq B_i$ and $\cup_{i\in I} A_i \subseteq \cup_{i\in I} B_i$, so $f \in \prod_{i\in I} B_i$. $\square$

For the power set $D = 2^{(\mathrm{Loc}^n)}$ of all tuples of length n, ordered by inclusion, Y the set of all Cartesian products in D, and the set of singletons as a generating subset the assumption is satisfied: singletons are Cartesian products and the union of a chain of Cartesian products is a Cartesian product.

Combinatorial Problem: If the underlying set Loc is finite, how large can a maximized antichain of Cartesian products be?

We don't know how to solve this problem at the moment.

For a set of tuples $A \subseteq \mathrm{Loc}^n$, $i \in \mathbb{N}_n$ and $r \in \mathrm{Loc}$ let us call

$$A_{i,r} = \pi_{\mathbb{N}_n \setminus \{i\}} \{a \in A \mid a_i = r\}$$

a *restriction* of A (with parameters i, r). An $(n-1)$-tuple lies in this set exactly if, whenever r would be inserted at the ith position, the tuple would lie in A. Since projection is monotonic, restrictions are monotonic also, i.e.

$$\forall\, A \subseteq B \subseteq \mathrm{Loc}^n, i \in \mathbb{N}_n, r \in \mathrm{Loc} : \; A_{i,r} \subseteq B_{i,r} \, .$$

Lemma 18 (AC). *Let a set $A \subseteq \mathrm{Loc}^n$ be represented as a maximized antichain M of Cartesian products. Then for each $i \in \mathbb{N}_n, r \in \mathrm{Loc}$, the set $A_{i,r}$ has a representation as a union of a maximized antichain M' of Cartesian products with no greater cardinality than $|M|$. If* Loc *is finite and Cartesian products are stored componentwise, the elements of the new maximized antichain can be computed in polynomial time in n, $|\mathrm{Loc}|$ and $|M|$.*

Proof. Let M be the maximized antichain of Cartesian products with union A and let us fix $i \in \mathbb{N}_n, r \in \mathrm{Loc}$. Let $M_{i,r} = \{C_{i,r} \mid C \in M\}$. First we show that its union is $\bigcup M_{i,r} = A_{i,r}$.

"$\subseteq$": Each $C \in M$ is contained in A, so $C_{i,r} \subseteq A_{i,r}$ by monotony of restrictions. Thus $\bigcup_{C \in M} C_{i,r} \subseteq A$.

"$\supseteq$": If $a \in A_{i,r}$ then $b := a \cup \{(i,r)\} \in A$ and there is some $C \in M$ with $b \in C$. Then $a \in C_{i,r} \subseteq \bigcup M_{i,r}$.

To prove that $M_{i,r}$ is maximized, let $K = \prod_{1 \le j \le n, j \ne i} K_j$ be a Cartesian product of $(n-1)$-tuples contained in $A_{i,r}$ with components $K_j \subseteq \mathrm{Loc}$ $(1 \le j \le n, j \ne i)$ and let $\tilde{K} = \{l \in \mathrm{Loc}^n \mid \exists\, a \in K : a \cup \{(i,r)\} = l\}$ so that $(\tilde{K})_{i,r} = K$. Then $\tilde{K} = \{f : \mathbb{N}_n \to \mathrm{Loc} \mid f(i) \in \{r\} \text{ and } \forall\, j \ne i : \; f(j) \in K_j\}$ is a Cartesian product in D and is thus contained in some maximal Cartesian product $C \in M$ as a subset. Restrictions preserve inclusions, so $K = (\tilde{K})_{i,r} \subseteq C_{i,r}$. So the set $M_{i,r}$ is maximized.

In case Loc is infinite, Prop. 16 says there is a subset M' of $M_{i,r}$ which is a maximized antichain with the same union $A_{i,r}$. Since there is a bijection between M and $M_{i,r} \supseteq M'$, the set M' is at most as large as M.

In case Loc is finite, $M_{i,r}$ can be constructed from M in polynomial time. Then we set originally $M' := \emptyset$. For each Cartesian product $A \in M_{i,r}$ we check whether there is a strictly larger Cartesian product $B \in M_{i,r}$ and if no, we add A to M'. After at most $|M_{i,r}|^2$ many iterations all elements of M' are incomparable, their union is still $A_{i,r}$ and M' is maximized by construction. □

In the following, we reduce the problem of computing the abstract parameterized post to a simpler problem about the "standard" Cartesian abstraction and concretization maps:

$$\alpha_c : 2^{(\mathrm{Loc}^n)} \to (2^{\mathrm{Loc}})^n, S \mapsto (\pi_i S)_{i=1}^n \,,$$
$$\gamma_c : (2^{\mathrm{Loc}})^n \to 2^{(\mathrm{Loc}^n)}, T \mapsto \prod_{i=1}^n T_i \,.$$

We call the elements of $(2^{\mathrm{Loc}})^n$ *Cartesian abstract* elements. There is an injection of the set of Cartesian products in Loc^n into the set of Cartesian abstract elements. The nonempty Cartesian products get bijectively mapped to the tuple of their components, the empty Cartesian product has a nonunique representation as a tuple of sets among which at least one set is empty (for $n > 0$).

Assumption: For the rest of this section let the local store be finite. Each element of $(2^{\mathrm{Loc}})^n$ is represented as a list of n entries, each entry is itself a list of some elements from Loc.

Proposition 19. *The question whether a Cartesian product is a subset of a set of tuples can be solved in polynomial time.*

Formally: there is an algorithm that computes the map

$$2^{(\mathrm{Loc}^n)} \times (2^{\mathrm{Loc}})^n \to \mathrm{Bool},\ (E, A) \mapsto \gamma_c A \overset{?}{\subseteq} E$$

where E is represented as a set M of Cartesian abstract elements so that $\gamma_c M$ is a maximized antichain and $E = \bigcup \gamma_c M$, in polynomial time in $|M|$, n and $|\mathrm{Loc}|$.

Proof. The algorithm tests for each $B \in M$, whether $\gamma_c A \subseteq \gamma_c B$, and returns yes, if any of these tests succeeds, otherwise it returns no. The algorithm is correct, since $\gamma_c M$ is maximized. Since $\gamma_c A \subseteq \gamma_c B$ if and only if $((\forall\, i \in \mathbb{N}_n : A_i \subseteq B_i)$ or $\exists\, i \in \mathbb{N}_n : A_i = \emptyset)$, each such test is done in polynomial time in n and in Loc, so all the $|M|$ tests are done in polynomial time in n, $|M|$, and $|\mathrm{Loc}|$. □

Proposition 20. *The smallest Cartesian product that contains another Cartesian product without an exception set can be computed in polynomial time.*

Formally: there is an algorithm that computes the map

$$2^{(\mathrm{Loc}^n)} \times (2^{\mathrm{Loc}})^n \to (2^{\mathrm{Loc}})^n,\ (E, A) \mapsto \alpha_c(E^c \cap \gamma_c A)$$

where E is represented as a set M of Cartesian abstract elements so that $\gamma_c M$ is a maximized antichain and $E = \bigcup \gamma_c M$, in polynomial time in $|M|$, n and $|\mathrm{Loc}|$.

Proof. Let $E \subseteq \mathrm{Loc}^n$, $A \in (2^{\mathrm{Loc}})^n$. If $\gamma_c A$ is empty (which holds iff $A_i = \emptyset$ for some $i \in \mathbb{N}_n$), then the return value is the tuple of empty sets. Otherwise all A_i are nonempty.

Claim: For all $r \in \mathrm{Loc}, i \in \mathbb{N}_n$ holds:

$$r \in (\alpha_c(E^c \cap \gamma_c A))_i \quad \Leftrightarrow \quad r \in A_i \text{ and } \prod_{j \in \mathbb{N}_n \setminus \{i\}} A_j \not\subseteq E_{i,r} \,.$$

To prove the "$\Rightarrow$" direction, let $r \in (\alpha_c(E^c \cap \gamma_c A))_i = \pi_i(E^c \cap \gamma_c A)$. So there is an n-tuple $a \in E^c \cap \prod_{i=1}^n A_i$ with $a_i = r$, thus $r \in A_i$. Moreover $a \notin E$ and $a_j \in A_j$ ($j \in \mathbb{N}_n$). So the $(n-1)$-tuple $a \setminus \{(i,r)\} \in \prod_{j \in \mathbb{N}_n \setminus \{i\}} A_j$, but $a \setminus \{(i,r)\} \notin E_{i,r}$.

To prove "$\Leftarrow$", let $r \in A_i$ and let a be an $(n-1)$-tuple with $a \in \prod_{j \in \mathbb{N}_n \setminus \{i\}} A_j$ and $a \notin E_{i,r}$. Then $a \cup \{(i,r)\} \notin E$, but $a \cup \{(i,r)\} \in \prod_{j=1}^n A_j = \gamma_c A$. Thus $a \cup \{(i,r)\} \in E^c \cap \gamma_c A$, hence $r \in \pi_i(E^c \cap \gamma_c A) = (\alpha_c(E^c \cap \gamma_c A))_i$.

The claim is proven. By Lemma 18, for each $i \in \mathbb{N}_n, r \in \mathrm{Loc}$, there is a maximized antichain $M'_{i,r}$ of Cartesian products with union $E_{i,r}$ and componentwise representation of Cartesian products as Cartesian abstract elements, computed in polynomial time. Since $M'_{i,r}$ is maximized, $A' \subseteq M'_{i,r}$ if and only if $\exists\, C \in M'_{i,r} :\ A' \subseteq C$ for any Cartesian product A', especially for $\prod_{j \in \mathbb{N}_n \setminus \{i\}} A_j$. So for each $r \in \mathrm{Loc}, i \in \mathbb{N}_n$ holds:

$$r \in (\alpha_c(E^c \cap \gamma_c A))_i \qquad \Leftrightarrow \qquad r \in A_i \ \text{ and } \ \forall\, C \in M'_{i,r} : \prod_{j \in \mathbb{N}_n \setminus \{i\}} A_j \not\subseteq C\,.$$

Since $M'_{i,r}$ is generated in polynomial time and inclusion of Cartesian products is polynomial-time by Lemma 17, all the components of the abstract element $\alpha_c(E^c \cap \gamma_c A)$ are computable in polynomial time. $\square$

Now we go over to the domains used in program analysis, namely to $D = 2^{\mathrm{States}} = 2^{\mathrm{Glob} \times \mathrm{Loc}^n}$ and $D^\# = (2^{\mathrm{Glob} \times \mathrm{Loc}})^n$.

Proposition 21. *The smallest abstract element that is greater than or equal to the concretization of another abstract element without an exception set can be computed in polynomial time.*

Formally: Assume that for $E \in D$, each $\{l \mid (g,l) \in E\}$ (for $g \in \mathrm{Glob}$) is represented a set of Cartesian abstract elements whose concretizations form a maximized antichain and have $\{l \mid (g,l) \in E\}$ (for $g \in \mathrm{Glob}$) as a union. Then computing the map

$$D \times D^\# \to D^\#, \quad (E, A) \mapsto \alpha_{\mathrm{cart}}(E^c \cap \gamma_{\mathrm{cart}} A)$$

needs polynomial time in n, $|\mathrm{Loc}|$, $|\mathrm{Glob}|$ and the maximum cardinality of an antichain.

Proof. Let $A \in D^\#$ and $E \in D$. For each $g \in \mathrm{Glob}$ and $i \in \mathbb{N}_n$ let $A_i^{[g]} := \{l \mid (g,l) \in A_i\}$ and $A^{[g]} := \prod_{i \in \mathbb{N}_n} A_i^{[g]}$. For all $g \in \mathrm{Glob}$, $l \in \mathrm{Loc}^n$ holds: $((g,l) \in \gamma_{\mathrm{cart}} A)$ iff $(\forall\, i \in \mathbb{N}_n : (g, l_i) \in A_i)$ iff $(\forall\, i \in \mathbb{N}_n : l_i \in A_i^{[g]})$ iff $((g,l) \in \{g\} \times \prod_{i=1}^n A_i^{[g]} = \{g\} \times A^{[g]})$. Thus

$$\gamma_{\mathrm{cart}} A = \bigcup_{g \in \mathrm{Glob}} \left(\{g\} \times A^{[g]}\right). \tag{1}$$

For $g \in \mathrm{Glob}$, let $E^{(g)} = \{l \mid (g,l) \in E\}$. For any $g \in \mathrm{Glob}$ and $B \subseteq \mathrm{Loc}^n$ holds:

$$(\{g\} \times B) \setminus E = \{g\} \times \left(B \setminus E^{(g)}\right). \tag{2}$$

The map

$$\beta : \mathrm{Glob} \times \left(2^{\mathrm{Loc}}\right)^n \to D^{\#}, \quad (g, (B_i)_{i=1}^n) \mapsto (\{g\} \times B_i)_{i=1}^n$$

makes abstract elements from Cartesian abstract elements and is computable in polynomial time. For any $B \subseteq \mathrm{Loc}^n$ holds:

$$\alpha_{\mathrm{cart}}(\{g\} \times B) = (\{(g, l_i) \mid l \in B\})_{i=1}^n = (\{g\} \times \pi_i B)_{i=1}^n = \beta(g, \alpha_c B)\,. \qquad (3)$$

Now

$$\alpha_{\mathrm{cart}}(E^c \cap \gamma_{\mathrm{cart}} A) \overset{(1)}{=} \alpha_{\mathrm{cart}}\left(E^c \cap \bigcup_{g \in \mathrm{Glob}} \left(\{g\} \times A^{[g]}\right)\right) = [\text{distributivity}]$$

$$\alpha_{\mathrm{cart}}\left(\bigcup_{g \in \mathrm{Glob}} \left(\left(\{g\} \times A^{[g]}\right) \setminus E\right)\right) = [\text{abstraction map is a join-morphism}]$$

$$\bigsqcup_{g \in \mathrm{Glob}} \alpha_{\mathrm{cart}}\left(\left(\{g\} \times A^{[g]}\right) \setminus E\right) \overset{(2)}{=} \bigsqcup_{g \in \mathrm{Glob}} \alpha_{\mathrm{cart}}\left(\{g\} \times \left(A^{[g]} \setminus E^{(g)}\right)\right) \overset{(3)}{=}$$

$$\bigsqcup_{g \in \mathrm{Glob}} \beta\left(g, \alpha_c\left(\left(\gamma_c\left(A_i^{[g]}\right)_{i=1}^n\right) \setminus E^{(g)}\right)\right).$$

From Prop. 20 we know that $\alpha_c\left(\left(\gamma_c\left(A_i^{[g]}\right)_{i=1}^n\right) \setminus E^{(g)}\right)$ is computable in polynomial time in n and $|\mathrm{Loc}|$ and the maximum cardinality of an antichain; the map β is also polynomial and the abstract join is also polynomial. □

Proposition 22. *Computing the best abstract post with exceptions takes polynomial time.*

Formally: Assume that for $E \in D$, each $\{l \mid (g, l) \in E\}$ (for $g \in \mathrm{Glob}$) is represented as a set of Cartesian abstract elements whose concretizations form a maximized antichain and have $\{l \mid (g, l) \in E\}$ as the union. Then computing the map

$$D \times D^{\#} \to D^{\#}, (E, A) \mapsto \mathrm{post}_{E,\mathrm{cart}} A$$

takes polynomial time in n, $|\mathrm{Loc}|$, $|\mathrm{Glob}|$ and the maximum size of an antichain from the representation of E.

Proof. Let $A := (A_i)_{i=1}^n \in D^{\#+}$. Proceeding as in the proof of Prop. 8 we obtain $\alpha_{\mathrm{cart}} \alpha_E \mathrm{post} \gamma_E \gamma_{\mathrm{cart}} A = \alpha_{\mathrm{cart}}(E^c \cap \mathrm{post} E) \sqcup \alpha_{\mathrm{cart}}(E^c \cap \mathrm{post} F \gamma_{\mathrm{cart}} A)$.

In section 4.1 we have shown that $F \gamma_{\mathrm{cart}} A = \bigcup \gamma(\bar{F} A)$ where $\bar{F} A \subseteq D^{\#+}$ has polynomially many elements. So $\alpha_{\mathrm{cart}}(E^c \cap \mathrm{post} F \gamma_{\mathrm{cart}} A) = \alpha_{\mathrm{cart}}(\bigcup \{E^c \cap \gamma_{\mathrm{cart}} y \mid y \in \bar{F} A\}) = \bigsqcup \{\alpha_{\mathrm{cart}}(E^c \cap \gamma_{\mathrm{cart}} y) \mid y \in \bar{F} A\}$, which can be computed in polynomial time by the assumption and Prop. 21.

Now let the representation of E be $E = \cup_{g \in \mathrm{Glob}}(\{g\} \times \bigcup \gamma_c M_g)$ where M_g is a set of Cartesian abstract elements so that $\gamma_c M_g$ is a maximized antichain and $\bigcup \gamma_c M_g = \{l \mid (g, l) \in E\}$. Then $\alpha_{\mathrm{cart}}(E^c \cap \mathrm{post} E) = \alpha_{\mathrm{cart}}(E^c \cap$

$\cup_{g\in\mathrm{Glob}}\mathrm{post}(\{g\}\times\bigcup\gamma_c M_g)) = \bigsqcup_{g\in\mathrm{Glob},C\in M_g}\alpha_{\mathrm{cart}}(E^c\cap\mathrm{post}(\{g\}\times\gamma_c C))$. Repeating the argument from section 4.1 again, we obtain that $\mathrm{post}(\{g\}\times\gamma_c C)$ is a union of polynomially many elements of D^+. Applying assumption and Prop. 21 completes the proof. □

Corollary 23. *Computing the least abstract fixpoint with exceptional Cartesian abstraction and representation of E so that each $\{l \mid (g,l)\in\mathrm{Loc}\}$ is a union of Cartesian products needs polynomial time.*

Formally: Assume that for $E\in D$, each $\{l \mid (g,l)\in E\}$ (for $g\in\mathrm{Glob}$) is represented as a set of Cartesian abstract elements whose concretizations form a maximized antichain and have $\{l \mid (g,l)\in E\}$ as the union. Then computing the map

$$D\times D\to D^{\#},\quad (E,\mathrm{init})\mapsto \mathrm{lfp}\,(\lambda X.\alpha_{\mathrm{cart}}\alpha_E(\mathrm{init}\cup\mathrm{post}\gamma_E\gamma_{\mathrm{cart}}X))$$

needs polynomial time in n, $|\mathrm{Loc}|$, $|\mathrm{Glob}|$, in the cardinality of the largest antichain and in $|\mathrm{init}|$.

Proof. The computation creates elements $X^0=\bot$, and $X^{i+1}=\alpha_{\mathrm{cart}}\alpha_E\mathrm{init}\sqcup\mathrm{post}_{E,\mathrm{cart}}X^i$. The part $\alpha_{\mathrm{cart}}\alpha_E\mathrm{init}$ is constant, the part $\mathrm{post}_{E,\mathrm{cart}}X^i$ takes polynomial time to compute by Prop. 22, so each iteration needs polynomial time. The chain length of the abstract domain is $n|\mathrm{Glob}||\mathrm{Loc}|+1$ which makes the number of iterations also polynomial. □

Remark that if initial states are represented the same way as the exception set then the run time is polynomial in the cardinality of the largest antichain from the representation of init instead of $|\mathrm{init}|$.

4.4 Low space and low time?

After having seen three different algorithms we ask whether there is an algorithm that needs at most polynomial space (in n, $|\mathrm{Loc}|$, $|\mathrm{Glob}|$ and $|E|$) for the exception set and at most polynomial time (in n, $|\mathrm{Loc}|$ and $|\mathrm{Glob}|$). Remembering Prop. 19, we believe that the core of the question lies in the following

Open problem: Is there a unary polynomial p over integers so that for all natural numbers n,d and for each subset $B\subseteq(\mathbb{N}_d)^n$ there is an algorithm (a deterministic Turing machine or a deterministic RAM program with a data structure, with logarithmic cost measure and built-in logarithmic cost addition) of size $\le p(d)^{p(n)}$ that, in time $\le p(dn)$, given an n-tuple $(A_i)_{i=1}^n$ of subsets of $\mathbb{N}_d$ as an input, answers the question whether $\prod_{i=1}^n A_i\subseteq B$?

5 Examples

Now we present a couple of examples. For better understanding we perform the computation by definition of exceptional abstraction and not along the lines of the polynomial-time algorithms.

5.1 First Thread Waits

Let's try to prove that the label D never gets reached in the following program:

Global variable g=0

```
Thread 1:          ||Thread 2:
A: wait until g=1; ||E: g:=1;
B: wait until g=0; ||F: g:=0;
C: wait until g=1; ||G:
D:                 ||
```

In the definition of a multithreaded program we required for simplicity that local stores of all threads are equal, so for formal reasons we let the local variable be the program counter pc taking values from $\{A, B, C, D, E, F, G\}$.

Choosing the empty exception set $E = \emptyset$ leads to $A = \text{lfp}\,(\lambda X.\alpha_{\text{cart}}(\text{init} \cup \text{post}\gamma_{\text{cart}}X)) = (\{0A, 1A, 0B, 1B, 0C, 1C, 0D, 1D\}, \{0E, 1F, 0G\})$, whose concretization $\gamma_{\text{cart}}A$ contains D as a local component of the first thread. Here, the simplified notation XY (e.g. $0A$) means the pair of valuation maps $(\{(g, X)\}, \{(pc, Y)\})$. The precision of the Flanagan-Qadeer algorithm (which is equivalent to our approach with the empty exception set) is insufficient.

For comparison we choose the exception set $E = \{0CG, 0BG\}$. (It is a shorthand for $\{(\{(g, 0)\}, \{(pc, C)\}, \{(pc, G)\}), (\{(g, 0)\}, \{(pc, B)\}, \{(pc, G)\})\}$, viewing each state as a tuple of valuation maps.) The initial state is $0AE$.

The fixpoint iteration sequence is $X^0 = (\emptyset, \emptyset)$ and $X^{i+1} = \alpha_{\text{cart}}\alpha_E\text{init} \sqcup \alpha_{\text{cart}}((\text{post}(E \cup \gamma_{\text{cart}}X^i)) \setminus E)$ for $i \geq 0$:

$$
\begin{aligned}
X^0 &= (\emptyset, \emptyset)\\
X^1 &= (\{0A\}, \{0E\}) \sqcup \alpha_{\text{cart}}\underbrace{((\text{post}\{0CG, 0BG\}) \setminus \{0CG, 0BG\})}_{\emptyset} = (\{0A\}, \{0E\})\\
X^2 &= (\{0A\}, \{0E\}) \sqcup \alpha_{\text{cart}}\underbrace{((\text{post}\{0BG, 0CG, 0AE\}) \setminus \{OBG, OCG\})}_{\{1AF\}}\\
&= (\{0A, 1A\}, \{0E, 1F\})\\
X^3 &= (\{0A\}, \{0E\}) \sqcup \alpha_{\text{cart}}\underbrace{((\text{post}\{0BG, 0CG, 0AE, 1AF\}) \setminus \{OBG, OCG\})}_{\{1AF, 1BF, 0AG\}}\\
&= (\{0A, 1A, 1B\}, \{0E, 0G, 1F\})\\
X^4 &= (\{0A\}, \{0E\}) \sqcup \alpha_{\text{cart}}\underbrace{((\text{post}\{0BG, 0CG, 0AE, 0AG, 1AF, 1BF\}) \setminus \{OBG, OCG\})}_{\{1AF, 1BF, 0AG\}}\\
&= (\{0A, 1A, 1B\}, \{0E, 0G, 1F\})
\end{aligned}
$$

$$\gamma_E\gamma_{\text{cart}}X^3 = \{0BG, 0CG, 0AE, 0AG, 1AF, 1BF\}\,.$$

The concretization of the fixpoint doesn't contain D as a local state of the first thread, which is what we wanted to prove.

5.2 Mutual Exclusion for One Lock

We give an important large class of multithreaded programs where mutual exclusion can be proven in polynomial time using polynomial space for the exception set.

Consider a multithreaded program with n threads and global variables, one of which is called m, initially 0. Each thread looks like follows:

...
A: acquire m;
... critical section ...
B: release m;
...

where the program satisfies the following criteria:

1. Initial states don't violate mutual exclusion, i.e. for any initial state there is at most one thread inside its critical section between the locations A and B;
2. The only jumps from inside the critical section to outside or vice versa are the acquire and release statements at A and B;
3. The only transition that sets m to zero is "B: release m".

Statements at program locations other than A or B might vary from thread to thread, include jumps or be missing completely. The statement "acquire m" atomically waits till $m = 0$ and sets it to some nonzero value. The statement "release m" sets m to zero.

Let $(A, B] := \{l \in \mathrm{Loc} \mid A < l(pc) \leq B\}$ and $(A, B]^c = \mathrm{Loc} \setminus (A, B]$ its complement. Further for $1 \leq i \leq n$ let $C(i) = ((A, B]^c)^{i-1} \times (A, B] \times ((A, B]^c)^{n-i}$.

Proposition 24. *The set $M = \{C(i) \mid i \in \mathbb{N}_n\}$ is a maximized antichain.*

Proof. Let $i, j \in \mathbb{N}_n$ be different and $a \in C(i) \cap C(j)$. Then $a_i \in (A, B] \cap (A, B]^c = \emptyset$. So each two Cartesian products in M are disjoint, especially M is an antichain.

Now we show that M is maximized. Let $H = \prod_{i=1}^n H_i \subseteq \bigcup M$. Assume for the purpose of contradiction that there is no Cartesian product in M that contains H completely. Since H is contained in the union of M, there should be at least two Cartesian products in M that intersect H, say, $H \cap C(i) \neq \emptyset \neq H \cap C(j)$ for $i, j \in \mathbb{N}_n$ and $i \neq j$. Then there are tuples $a \in H \cap C(i)$ and $b \in H \cap C(j)$. We have $a_i \in (A, B]$, $b_j \in (A, B]$ and $b_j \in H_j$. Let $\tilde{a} = (a \setminus \{(j, a_j)\}) \cup \{(j, b_j)\}$. Then $\tilde{a} \in H$. But $(\tilde{a})_i \in (A, B]$ and $(\tilde{a})_j \in (A, B]$ and $i \neq j$, so $\tilde{a} \notin C(k)$ for any $k \in \mathbb{N}_n$, in contradiction to $H \subseteq \bigcup_{k \in \mathbb{N}_n} C(k)$. □

We show that choosing $E = \cup_{g \in \mathrm{Glob}, g(m) \neq 0, C \in M} \{g\} \times C$ as an exception set suffices to prove mutual exclusion. Let $R = E \cup (\mathrm{Glob} \times ((A, B]^c)^n)$.

Proposition 25. *R is an inductive program invariant, i.e.* init $\subseteq R$ *and* post$R \subseteq R$.

Proof. By criterion 1 each initial state has either no thread in critical section or at most one thread in critical section. So init $\subseteq (\mathrm{Glob} \times ((A, B]^c)^n) \cup E = R$. Now let $(g, l) \in R$ and $(g, l) \to_i (g', l')$. There are several cases.

– Case $g(m) = 0$, $l \in ((A, B]^c)^n$, $l_i(pc) = A$. Then by definition of acquire $g' \neq 0$ and $l' \in (A, B]$, so $(g', l') \in E$.
– Case $g(m) \neq 0$, $l \in ((A, B]^c)^n$, $l_i(pc) = A$. Then transition "acquire m" cannot fire, especially (g', l') cannot occur due to a step of the ith thread.
– Case $l \in ((A, B]^c)^n$, $l_i(pc) \neq A$. From criterion 2 follows $l'_i \notin (A, B]$, so $(g', l') \in \text{Glob} \times ((A, B]^c)^n$.
– Case $g(m) \neq 0$ and $l \in C(i) \in M$, $l'_i \in (A, B]$. Then $g'(m) \neq 0$ by criterion 3 and $(g', l') \in E$.
– Case $g(m) \neq 0$ and $l \in C(i) \in M$, $l'_i \in (A, B]^c$. Then $(g', l') \in \text{Glob} \times ((A, B]^c)^n$.
– Case $g(m) \neq 0$ and $l \in C(j) \in M$, $j \neq i$, $l'_i \in (A, B]$. From $l \in C(j)$ follows $l_i \in (A, B]^c$. From criterion 2 follows $l_i(pc) = A$ and the transition is "acquire m". But this contradicts to $g(m) \neq 0$.
– Case $g(m) \neq 0$ and $l \in C(j) \in M$, $j \neq i$, $l'_i \in (A, B]^c$. Then $l_i \in (A, B]^c$ and criterion 3 implies $g'(m) \neq 0$. So $l' \in C(j)$ and $(g', l') \in E$. □

Notice that for $Y := (\text{Glob}\times(A, B]^c)_{i=1}^n$ holds $R = \gamma_E \gamma Y$, so applying Prop. 7 to R, Y and $F = \text{init}\cup\text{post}$ gives $\gamma_E\gamma_{\text{cart}}(\text{lfp}\ (\lambda Z.\alpha_{\text{cart}}\alpha_E(\text{init}\cup\text{post}\gamma_E\gamma_{\text{cart}}Z))) \subseteq R$. Since R shows mutual exclusion and the set generated by the parameterized fixpoint computation is a subset of R, mutual exclusion is proven by abstract fixpoint checking parameterized with E. Moreover, all antichains in the representation of E have linear cardinality in n, so the algorithm from section 4.3 run on the "maximized antichains"-representation consumes polynomial time and space. No state explosion occurs.

5.3 Mutual Exclusion for Many Locks

We show how does the exception set look like for programs with a variable number of threads and a variable number of lock variables.

We describe now a class of multithreaded program with n threads and k locks. For simplicity, we number the lock variables by natural numbers. Let $\text{Glob} = 2^{\mathbb{N}_k}$ be the set of global stores. If a set $G \in \text{Glob}$ is a part of a program state, then in this state exactly the locks in G are held. Let $\text{Loc} = \mathbb{N}_L$ be the set of control locations of each thread. For simplicity, we assume no other local variables except the program counters and no other global variables except locks. The only allowed statements for any thread i are
– "j: acquire lck; $j+1$:", representing all transitions $(g, j) \to_i (g \cup \{\text{lck}\}, j+1)$ where $\text{lck} \notin g$;
– "j: release lck; $j+1$:"', representing all transitions $(g, j) \to_i (g \setminus \{\text{lck}\}, j+1)$;
– "j: if (ϕ) goto j'; $j + 1$:", representing all transitions $(g, j) \to_i (g, j')$ where (g, j) satisfies the formula ϕ and all transitions $(g, j) \to_i (g, j + 1)$ where (g, j) doesn't satisfy ϕ.

Given the program text of a thread, a *critical section* is a contigious sequence of control locations that starts right after a statement "acquire lck" and ends just before the next following "release lck" statement, for the same lock variable lck. If there is no following "release lck" statement, the critical section ends at the end of the thread. The lock variable "lck" is said to *protect* all the locations in this critical section. A program state has the *mutual exclusion* property if in this state, for every lock, at most one thread is in any of its critical sections protected by this lock. The whole program has the *mutual exclusion* property if there is no execution that starts in an initial state and ends in a state that violates mutual exclusion. The programs, for which we would like to derive the exception set, should satisfy the following criteria:

1. For any initial state, if a lock variable lck is held, then there is exactly one thread inside some of its critical sections protected by lck, and if it's not held, there is no thread inside any of its critical sections protected by lck.
2. The only jumps from inside a critical section to outside or vice versa is via the acquire and release statements.
3. Within any thread, for any $lck \in \mathbb{N}_k$ and statement "release lck", there is a preceding statement "acquire lck" without other "release lck" in between.

An example of a thread of such a program is

1: acquire lck_1;
2: acquire lck_2;
3: release lck_1;
4: if (lck_1) goto 7;
5: acquire lck_3;
6: release lck_3;
7:

The description of the program class is finished, now we are going to define the exception set for such a program.

An *n-partition* of a set X is an tuple of n disjoint subsets of X with union X. For $G \in \text{Glob}$, let $P_n(G)$ be the set of n-partitions of G. For $G \in \text{Glob}$ and $1 \leq i \leq n$, let $C(G, i) = \{j \in \text{Loc} \mid \forall\, lck \in \mathbb{N}_k : lck \text{ protects } j \text{ in thread } i \text{ iff } lck \in G\}$ be the set of control locations of thread i which are protected by exactly all the locks in G. For each n-partition $S = (S_i)_{1 \leq i \leq n}$ of a global store, let $C(S) = \prod_{i=1}^{n} C(S_i, i)$. For each $G \in \text{Glob}$, let $M(G) = \{C(S) \mid S \in P_n(G)\}$ and $E(G) = \bigcup M(G)$. Let the exception set be defined as $E = \bigcup_{G \in \text{Glob}, G \neq \emptyset} \{G\} \times E(G)$. Let $N = \{\emptyset\} \times E(\emptyset)$ and $R = E \cup N$.

Lemma 26. *The set R is an inductive invariant.*

Proof. By criterion 1 each initial state $(G, l) \in \text{init}$ has either no thread in critical section or, for each lock, there is at most one thread in a critical section protected by this lock. In the first case $(G, l) \in \{\emptyset\} \times \prod_{i=1}^{n} C(\emptyset, i) = \{\emptyset\} \times C(\emptyset) = N \subseteq R$. In the second case let $P(i) = \{lck \in G \mid lck \text{ protects } l_i \text{ in thread } i\}$ for all $1 \leq i \leq n$. If we have $P(i) \cap P(j) \neq \emptyset$ for some $1 \leq i, j \leq n$, $i \neq j$, then some $lck \in G$ would protect both l_i and l_j in threads i and j, respectively, in contradiction to criterion 1. So $P(i)$ and $P(j)$ are disjoint for different $1 \leq i, j \leq n$. By the same criterion, each $lck \in G$ protects a critical section in some thread $1 \leq i \leq n$, implying $lck \in P(i)$. So $(P(i))_{i=1}^{n}$ is a n-partition of G. Now let $1 \leq i \leq n$. For all $lck \notin G$, neither $lck \in P(i)$, nor lck protects l_i in thread i. For all $lck \in G$, the fact $lck \in P(i)$ is equivalent to lck protecting l_i in thread i. So for all $lck \in \mathbb{N}_n$, the facts $lck \in P(i)$ and lck protects l_i in thread i are equivalent. This proves $l_i \in C(P(i), i)$. So $l \in C((P(i))_{i=1}^n) \subseteq E(G)$. Thus $(G, l) \in E$. We have shown $\text{init} \subseteq R$.

Now let $(G, l) \in R$ with a successor (G', l'). Let i be the thread that contains the statement τ that induced the transition. Let S be a n-partition of G with $l \in C(S)$. Case split on τ.

– "l_i: acquire lck; l'_i:". So $lck \notin G$ and $G' = G \cup \{lck\}$. Let $S'_i = S_i \mathbin{\dot\cup} \{lck\}$ and $S'_j = S_j$ for $j \neq i$. Then $S' = (S_j)_{j=1}^n$ is a partition of G'. We have $l_i \in C(S_i, i)$, so for all $lck' \in \mathbb{N}_k \setminus \{lck\}$, we have that lck' protects l_i in thread i if and only if $lck' \in S_i$. Thus for all $lck' \in \mathbb{N}_k \setminus \{lck\}$, we have that lck' protects l'_i in thread i if and only if $lck' \in S'_i$. We also have that lck' protects l'_i in thread i and $lck' \in S'_i$. So l'_i in $C(S'_i, i)$. Knowing that $l'_j = l_j \in C(S_j, j) = C(S'_j, j)$ for $j \neq i$, we have $l' \in E(G')$ and $(G', l') \in E$.
– "l_i: release lck; l'_i:". So $G' = G \setminus \{lck\}$. Let $S'_i = S_i \setminus \{lck\}$ and $S'_j = S_j$ for $j \neq i$. If $lck \in S_i$ then the tuple $S' = (S'_j)_{j=1}^n$ is a partition of G'. If $lck \in S_j$ for some $j \neq i$, then $lck \notin S_i$, so $l_i \in C(S_i, i)$ implies that lck doesn't protect l_i in thread i, so there is no critical section for lck ending at l_i, thus "release lck" at l_i has no preceding "acquire lck", contradicting criterion 3. If $lck \notin S_j$ for any $j \in \mathbb{N}_n$, then $S' = S$ is a partition of $G' = G$. So in both cases $lck \in G$ and $lck \notin G$ the tuple S' is a partition of G'. We have $l_i \in C(S_i, i)$, so for all $lck' \in \mathbb{N}_k \setminus \{lck\}$ we have that lck' protects l_i in thread i if and only if $lck' \in S_i$. Thus for all $lck' \in \mathbb{N}_k \setminus \{lck\}$, we have that lck' protects l'_i in thread i if and only if $lck' \in S'_i$. Also neither lck protects l'_i, nor $lck \in S'_i$. So $l'_i \in C(S'_i, i)$. Knowing that $l'_j = l_j \in C(S_j, j) = C(S'_j, j)$ for $j \neq i$, we have $l' \in E(G')$ and $(G', l') \in R$.
– "l_i: if(ϕ) goto b; c:". So $G = G'$. By criterion 2, no critical section is quit or entered by the jump statement. So we have "$\forall\, lck \in \mathbb{N}_k : lck$ protects l_i in thread i" if and only if "$\forall\, lck \in \mathbb{N}_k : lck$ protects b in thread i" if and only if "$\forall\, lck \in \mathbb{N}_k : lck$ protects c in thread i". Thus $l_i \in C(S_i, i)$ implies $b, c \in C(S_i, i)$. Knowing that $l'_i \in \{b, c\}$ and $l'_j = l_j$ for $j \neq i$, we have $l' \in C(S)$ and thus $(G', l') \in R$. $\square$

Lemma 27. *All states in R satisfy the mutual exclusion property.*

Proof. Let $(G, l) \in R$. Then $l \in C(S)$ for some n-partition of G, so $l_i \in C(S_i, i)$ for all $1 \leq i \leq n$. Let $lck \in \mathbb{N}_k$ and $i \neq j$ be two threads. If $lck \notin G$, then $lck \notin S_i \cup S_j$, so, by definition of C, the lock lck protects neither l_i nor l_j. If lck is in one of two sets S_i, S_j, say, in S_i, then $lck \notin S_j$, so lck protects l_i, but not l_j by definition of C. $\square$

Notice that for $Y = (\{\emptyset\} \times C(\emptyset, i))_{i=1}^n$ we have $\gamma_{\text{cart}} Y = N$, so applying Proposition 7 to $F = \lambda x.\, \text{init} \cup \text{post}\, x$, $R = \gamma_E \gamma Y$ gives us $\gamma_E \gamma_{\text{cart}}(\text{lfp}\,(\lambda x.\, \alpha_{\text{cart}} \alpha_E(\text{init} \cup \text{post} \gamma_E \gamma_{\text{cart}} x))) \subseteq R$. Thus doing thread-modular verification with exception set E proves mutual exclusion.

Lemma 28. *For every $G \in \text{Glob}$, the set $M(G)$ is a maximized antichain of Cartesian products.*

Proof. To show that $M(G)$ is an antichain, let S and S' be two different n-partitions of G. Then there is some component $1 \leq i \leq n$ so that $S_i \neq S'_i$. Without loss of generality, let $lck \in S_i \setminus S'_i$ (if no such lck exists, swap S and S'). Assume that $j \in C(S_i, i) \cap C(S'_i, i)$. Since $j \in C(S_i, i)$ and $lck \in S_i$, the lock lck protects j in thread i. Since $j \in C(S'_i, i)$ and $lck \notin S'_i$, the lock lck doesn't protect j in thread i. This is a contradiction, so the assumption was false and $C(S_i, i) \cap C(S'_i, i) = \emptyset$, proving that $C(S)$ and $C(S')$ are incomparable with repect to product ordering.

To show that $M(G)$ is maximized, let's assume the contrary, namely that some Cartesian product $C = \prod_{i=1}^{n} C_i$ is a subset of $E(G)$, but not a subset of any $C(S)$ for an n-partition of G. So let C nontrivially intersect $C(S)$ and $C(S')$ for different $S, S' \in P_n(G)$. Let $a \in C \cap C(S)$ and $b \in C \cap C(S')$. There is a component $i \in \mathbb{N}_n$ in with $S_i \neq S'_i$. We have $a_i \in C(S_i, i)$, but $C(S_i, i)$ is disjoint from $C(S'_i, i)$, so $a_i \notin C(S'_i, i)$. But $b_i \in C(S'_i, i)$. Let $a' = (a \setminus \{(i, a_i)\}) \cup \{(i, b_i)\}$. Then $a'_i \in C_i$, so $a' \in C$. Knowing $S_i \neq S'_i$, there are two cases.

In one case, $S'_i \subsetneq S_i$. Let $d \in E(G)$. Let $lck \in \mathbb{N}_k$ be any lock that protects d_j in some thread $j \in \mathbb{N}_n$. There is a n-partition S'' of G with $d \in C(S'')$. Then $d_j \in C(S''_j, j)$. Thus $lck \in S''_j \subseteq G$. On the other hand, let $lck \in G$. There is a n-partition S'' of G with $d \in C(S'')$. There is a $1 \leq j \leq n$ with $lck \in S''_j$. We have $d_j \in C(S''_j, j)$. Then lck protects d_j in thread j. We have proven that the set of locks that protect components of $d \in E(G)$ is exactly G:

$$\{lck \in \mathbb{N}_k \mid \exists\, j \in \mathbb{N}_n : lck \text{ protects } d_j \text{ in thread } j\} = G\,.$$

Knowing $a \in C \subset E(G)$, the components of a are protected by totally $|G|$ locks. From $S'_i \subsetneq S_i$, $a'_i = b_i \in C(S'_i, i)$ and $a_i \in C(S_i, i)$, we follow that the set of locks that protect a'_i in thread i is strictly smaller than the set of locks that protect a_i in thread i. For other threads $j \neq i$, the set of locks that protect $a_j = a'_j$, is S_j, which is disjoint from S_i. So a' is protected by less locks than a, thus, using our claim, $a' \notin E(G)$. It's a contradiction to $a' \in C \subseteq E(G)$.

In another case $S'_i \setminus S_i \neq \emptyset$. Since S and S' both have G as a union, there is some component $j \neq i$ so that S_j contains at least one lock, say, lck, from $S'_i \setminus S_i$. So $a'_j = a_j \in C(S_j, j)$ is protected by lck. But $lck \in S'_i$ and $a'_i = b_i \in C(S'_i, j)$, so a'_i is protected by lck also. But any element $d \in E(G)$ has the property that for different components $\bar{i}, \bar{i}'$, the locations $d_{\bar{i}}$ and $d_{\bar{i}'}$ are protected by disjoint sets of locks. This property doesn't hold for a', so $a' \notin E(G)$, in contradiction to $a' \in C \subseteq E(G)$. $\square$

Lemma 29. *On a RAM with logarithmic cost measure, the representation of the exception set E as a list of antichains of Cartesian products, one antichain per global part, needs space $O(nL(n+1)^k)$.*

Proof. We have $|M(G)| = \sum_{k_1+\ldots+k_n=|G|} \binom{|G|}{k_1,\ldots,k_n} = n^{|G|}$. For storing a Cartesian product $C(S)$ for any partition S, we need Ln memory cells, storing each component as a bitvector, using one bit per addressed memory cell. The consumed space per Cartesian product is thus Ln. Storing $M(G)$ as an array needs

$|M(G)|Ln + \log(|M(G)|) = n^{|G|+1}L + |G| \log n$ space. All such arrays, for each $G \in \text{Glob} \setminus \{\emptyset\}$, stored again as an array, need the following space :

$$\sum_{G \in \text{Glob}, G \neq \emptyset} (n^{|G|+1}L + |G| \log n) + \log(\text{Glob} - 1) =$$

$$= nL \left(\sum_{\mathbb{N}_k \supseteq G \neq \emptyset} n^{|G|} + \log n \sum_{\emptyset \neq G \subseteq \mathbb{N}_k} |G| \right) + \log(2^k - 1) \leq$$

$$\leq nL((n+1)^k - 1 + (\log n)(2^k - 1)) + k \leq nL((n+1)^k + (\log n)^k 2^k) + k =$$

$$= nL((n+1)^k + (2 \log n)^k) + k \leq nL((n+1)^k + (n+1)^k) + k \leq 3nL(n+1)^k.$$

□

For a constant number of locks, and a variable number of threads, the exception set still needs polynomial space, and thus allows computing the abstract fixpoint in polynomial time in the number of threads.

6 Experiments and Implementation

6.1 Experiments

We have experimented with the algorithm from Corollary 23. We run it on the example 5.2, where each thread consists of an acquire and a release statement but nothing else. Computing the best invariant with the usual forward search needed several hours for 18 threads, with run time growing exponentially with base 3. Thread-modular algorithm needed half a minute for a program of 30 threads; its run time grew at most in cubic time. Given the absolutely unoptimized implementation, small efforts to keep it going and excellent run times, we expect that this method to have a much greater potential than it shows now.

6.2 Towards an Implementation

Now we talk about practical aspects of a serious (non-experimental) implementation.

The current proof of Thm. 9 relies on the use of decision diagrams, each of them has in general exponential size in the measure of the domain, since it can have up to 2^m leaves, for example if the exception element is the bottom of the lattice ($E = \emptyset$). To deal with this issue one can use the usual methods like generating each bit of the answer alone, using m diagrams instead of one, a diagram for a bit of answer. This enables a vast amount of sharing isomorphic subgraphs in the diagrams, e.g. for $E = \emptyset$ the m diagrams compute projections on each of m positions in a bitstring, so each of the diagrams would have linear number of nodes in the number of variables m, thus reducing the size of the data structure from exponential to polynomial.

Attention should also be payed to generating these binary decision diagrams. We suggest doing it on-the-fly, i.e. given the representation of the exception set as a list of tuples or as a set of Cartesian products, we suggest the low-space algorithm at the first iteration. Then the results of the involved computations of $\{0,1\}^m \to \{0,1\}^m$, $v \mapsto \phi\alpha_{\mathrm{cart}}((\gamma_{\mathrm{cart}}\phi^{-1}v) \setminus E)$ (where ϕ and ϕ^{-1} convert abstract elements to and from the binary representations) get cached in the decision diagrams, each of which allows three outcomes 0, 1 and "don't know". In the following iterations, first a lookup in these decision diagrams is done and then, if it returns "don't know", the low-space algorithm with the usual representation of the exception set is started. More caching is possible as follows. Notice that the above map is monotone wtih respect to product order. It's possible to derive the value of a monotone map $f : \{0,1\}^m \to \{0,1\}$ on some $x \in \{0,1\}^m$, if it's known that either $f(x') = 1$ for some $x' < x$ or $f(x') = 0$ for some $x' > x$. There is a representation of monotone maps, based on the results of Kleitmann and Markowsky on the Dedekind's problem, that allows doing poly-time queries with derivation of some (not all derivable) values.

7 Other directions

The parameter can be chosen arbitrarily for Cartesian abstraction. However, some values of the parameter give you a precise answer and some don't. In this article we don't address the issue of how to find a value of the parameter that suits for a specific verification task.

8 Related work

The term "Cartesian abstraction" is also known as "independent attribute method" for more than 25 years (see [10]) in program analysis. To the best of our knowledge, it is unknown in its current form in analysis of multithreaded programs.

Thread-modular verification, inspired by assume-guarantee reasoning, was suggested by Flanagan and Qadeer in [5]: they use the same model of multithreaded programs as we do. Their algorithm for proving safety was a starting point of our research.

Henziger and al. adapted the Flanagan-Qadeer algorithm to the BLAST verification tool (see [7]) by combining with abstraction refinement of the transition relation of threads.

In [11] the authors discovered the relationship between the thread-modular algorithm and Cartesian abstract interpretation in the special setting of multithreaded programs. They gave an abstract lattice and a Galois-connection so that the algorithm computation could be formulated as abstract fixpoint checking in the abstract lattice. Comparing that abstraction to the parameterized Cartesian abstraction, one can show that if the exception set is empty, then the parameterized Cartesian successor operator $\mathrm{post}^{\#}_{\emptyset,\mathrm{cart}}$ is exactly the abstract post from [11]. So parameterized Cartesian post is a parameterized version of Cartesian post. In [11] the authors also gave an approximation operator on the concrete lattice so that the Flanagan-Qadeer algorithm has an equivalent formulation in terms of fixpoint checking on the concrete lattice. Two polynomial-time techniques based solely on Cartesian abstraction were presented that increased the precision of the abstraction for some important classes of programs. But these techniques had fixed precision and thus remained incomplete. In contradiction, the parameterized Cartesian abstraction can show every provable non-reachability property by a judicious choice of the parameter.

Giacobazzi and Ranzato develop the theory of complete abstract interpretation in [6]. We would like to exploit connections to that theory, including connections to a recent work of Dave Schmidt [15].

9 Conclusion

We have examined the problem of proving non-reachability in multithreaded programs. We suggested an algorithm based on Cartesian abstraction whose precision can be altered by a parameter, namely by the exception set, which contains program states that are not going to be involved into Cartesian abstraction. The parameterized algorithm takes time which is polynomial in the number of threads, while the choice of parameter allows successor computation with arbitrary precision. We discovered a time-space tradeoff for the abstract fixpoint computation. The larger the exception set is (as long as it is included in the set of reachable states), the more precise the successor computation is going to be. In the limit case, if the set contains exactly the reachable states of the program, no overapproximation happens. We formalized our approach in the abstract interpretation framework. We showed a large class of programs that allow checking mutual exclusion in polynomial time and space. We discussed some aspects of possible practical implementation.

References

1. Blanchet, B., *Introduction to Abstract Interpretation*, 2002, lecture script, `http://bblanche.gitlabpages.inria.fr/absint.pdf`.
2. Cousot, P., Cousot, R., *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs By Construction or Approximation of Fixpoints*, fourth ACM symposium on principles of programming languages, 1977.
3. Cousot, P., Cousot, R., *Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation*, in Conference Record of FPCA '95 SIGPLAN/SIGARCH/WG2.8 Conference on Functional Programming and Computer Architecture, pp. 170–181, La Jolla, California, U.S.A., 25–28 June 1995, ACM Press, New York, U.S.A.
4. Cousot, P., *Partial Completeness of the Abstract Fixpoint checking*, SARA 2000, LNAI 1864, pp. 1–25, 2000.
5. Flanagan, C., and Qadeer, S., *Thread-Modular Model Checking*, in T.Ball and S.K. Rajamani (Eds.): SPIN 2003, LNCS 2648, pp. 213–224, 2003, Springer-Verlag Berlin Heidelberg 2003.

6. Giacobazzi, R., Ranzato, F., *Making Abstract Interpretations Complete*, JACM, 1997.
7. Henzinger, T. A., Jhala, R., Majumdar, R., Qadeer, S., *Thread-modular abstraction refinement*, 15th Conference on Computer-Aided Verification, 2003.
8. Holzmann, G., *The Model Checker Spin*, IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279–295.
9. Kozen, D., *Lower Bounds for Natural Proof Systems*, FOCS 1977, pp. 261–262.
10. Jones, N. D., Muchnick, S. S., *Complexity of Flow Analysis, Inductive Assertion Synthesis, and a Language Due to Dijkstra*, in Program Flow Analysis, theory and applications, Eds. Steven S. Muchnick and Neil D. Jones, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1981.
11. Malkis, A., Podelski, A., Rybalchenko, A., *Thread-Modular Verification and Cartesian Abstraction*, presented at the Thread Verification workshop, Seattle, 21–22 August 2006.
12. Malkis, A., Podelski, A., Rybalchenko, A., *Thread-Modular Verification Is Cartesian Abstract Interpretation*, 3rd International Colloquium on Theoretical Aspects of Computing, Tunis, Tunisia, 20–24 November 2006.
13. Mehlhorn, K., *Data Structures and Algorithms 1: Sorting and searching*, Springer-Verlag Berlin Heidelberg, 1984.
14. Oberschelp, A., *Allgemeine Mengenlehre*, Bibliographisches Institut & F.A. Brockhaus AG, Mannheim 1994
15. Schmidt, D., *Comparing completeness properties of static analyses and their logics*, APLAS'06, Springer LNCS 4279, pp. 183–199.
16. Tarski, A., *A Lattice-theoretical Fixpoint Theorem and Its Applications*, Pacific J. Math. 5 (1955), 285–309.