Thread-Modular Verification and Cartesian Abstraction

Alexander Malkis, Andreas Podelski, and Andrey Rybalchenko

Max-Planck Institut für Informatik, Saarbrücken
Albert-Ludwigs-Universität Freiburg
EPFL IC IIF MTC, Lausanne

Accepted in 2006, last updated in January 2024.

Abstract Verification of multithreaded programs is difficult. It requires reasoning about state spaces that grow exponentially in the number of concurrent threads. Successful verification techniques based on modular composition of overapproximations of thread behaviors have been designed for this task. These techniques have been traditionally described in assume-guarantee style, which does not admit reasoning about the abstraction properties of the involved compositional argument. Flanagan and Qadeer thread-modular algorithm is a characteristic representative of such techniques. In this paper, we investigate the formalization of this algorithm in the framework of abstract interpretation. We identify the abstraction that the algorithm implements; its definition involves Cartesian products of sets. Our result provides a basis for the systematic study of similar abstractions for dealing with the state explosion problem. As a first step in this direction, we obtain polynomial-time algorithms based on Cartesian abstraction that enjoy increased precision with respect to the Flanagan-Qadeer method. We limit the design space for future polynomial-time algorithms by providing a characterization of minimal precision increase that leads to loss of polynomial-time complexity.

1 Introduction

Verification of multithreaded software is an important and difficult task (e.g. [10], [11]). The number of states of a multithreaded program is exponential in the number of threads. Inherent theoretical restrictions (see [8]) complicate the verification task.

However, there are many successful algorithms and tools for different types of communication between components of a concurrent system. For instance, one can mention algorithms in the SPIN model-checker (see [7]). The main challenge is to design algorithms that reason about concurrent software in a modular way. Modularity allows to avoid the explicit construction of the global state space by considering each thread in isolation (see [5] and [6]).

Assume-guarantee reasoning (see e.g. [9]) offers a prominent approach to devise thread-modular algorithms. The behavior of each thread with respect to global

variables is described by its guarantee. We can view this guarantee as a transition system. Then we apply model-checking procedure on each thread, which is executed in the environment defined by the guarantees of other threads. Additionally we verify that each thread satisfies its own guarantee.
However, assume-guarantee reasoning does not provide an insight about the abstraction process involved. All that is known about precision loss during search in a thread's state space is that during discovering states of the thread, the behavior of all other threads is reduced to their action on global variables. Nothing was known about the loss of precision for the program as a whole. It was unclear whether or how is it possible to represent thread-modular reasoning in the standard framework of abstract interpretation.
As soon as the abstraction is identified, it provides additional insight into the algorithms. For instance, one could try to increase precision of the abstraction, to optimize the algorithms, to combine with other abstractions or add refinement. One could construct modifications of the abstraction, derive the corresponding algorithms and look at their runtime and precision.
We study the Flanagan-Qadeer algorithm for thread-modular verification. The distinguishing property of this algorithm lies in its low complexity. It is polynomial in the number of threads. The low complexity has its price: the algorithm is incomplete. The algorithm is also of the assume-guarantee type. Each computed guarantee is a set of pairs of valuations of unprimed and primed global variables (g, g'). While model-checking a thread, each time a thread state is discovered, we allow the global variables to be changed according to the guarantees of other threads. Each time the thread itself changes global variables from a discovered state, the corresponding pair of valuations of global variables before and after the step is added to the guarantee. Upon convergence, the environment assumptions are devised and the discovered states of each threads include the reachable thread states.
We would like to identify the abstraction used by the Flanagan - Qadeer algorithm to be able to reason about the algorithm. In particular, we would like to know how far one can push the Flanagan-Qadeer algorithm while still being polynomial in time and space.
In this paper we study the abstraction used in Flanagan-Qadeer algorithm and identify the boundary. Our result is that Flanagan-Qadeer algorithm implements Cartesian abstraction in special setting with threads (a so-called local Cartesian abstraction).
This insight allows us to find the "least modification" of the Flanagan-Qadeer algorithm that leads to the loss of polynomial complexity. Thus, the identification of the abstraction provides the insight into the algorithm itself.
Using this boundary, we overcome the inherent limitation of the algorithm, namely being too coarse for many programs of interest. We obtain polynomial-time algorithms that are far more precise, being able to prove absence of errors for a large class of programs on which the original algorithm says "don't know". The first method, so-called relaxed frontier search, gains as much precision as possible up to the identified boundary of the Flanagan-Qadeer algorithm. The

second method, quadratic thread-modular method, improves on proving mutual exclusion for a large class of multithreaded programs where the original approach failed. These algorithms are based solely on Cartesian abstraction and thus do not exploit the structure of the underlying transition relations of threads. Their potential is that they can be can still be combined with other optimization techniques on other levels. For example, on the level of the transition relation, consider abstraction refinement (see [6]). We also suggest constructing more precise polynomial-time algorithms based on Cartesian abstraction using a particular class of lattices.
On the side of the theory, local Cartesian abstraction formulation does not immediately provide a basis for complexity analysis, since the concrete domain (which is both the domain and the range of local Cartesian abstraction) admits exponentially long chains. Since local Cartesian abstraction is "the what" of Flanagan-Qadeer algorithm, we obtain a polynomial time algorithm for reachability under local Cartesian abstraction.
To our best knowledge, this work is the first attempt to develop the theory of Cartesian abstraction for multithreaded programs.

Outline of the paper:
First we define our program model.
Then we explain the algorithm. We provide a small example.
After that we define a concrete and an abstract domain and a corresponding Galois-connection that involve Cartesian products of sets. We state our first theorem saying how the output of the Flanagan-Qadeer algorithm is expressible in the standard framework of abstract interpretation. We prove this theorem.
Then we define local Cartesian abstraction of multithreaded programs. We state our second theorem which says that the algorithm implements local Cartesian abstraction: the output of the algorithm represents local Cartesian abstraction of the program. We demonstrate the theorem on our example and prove it in general.
After that, we give a boundary of the abstraction, i.e. a modification with a minimal precision increase that immediately breaks the polynomial-time border of the Flanagan-Qadeer algorithm.
At last we look at the boundary, find two different sources of precision loss of the original algorithm and show them on examples. Then we describe the relaxed frontier search and quadratic thread-modular algorithms that overcome these sources of precision loss. We provide examples and discuss possible improvements based on the Cartesian abstraction approach.
We conclude by summarizing the work.

2 Preliminaries

2.1 Programs with Threads

We are interested in proving safety properties of multithreaded programs. Each safety property can be encoded as a reachability property.

For simplicity we consider programs consisting of only two threads (the general case can be readily devised). A two-threaded *program* is given by a tuple

$$(\mathrm{Glob}, \mathrm{Loc}_1, \mathrm{Loc}_2, \rightarrow_1, \rightarrow_2, \mathrm{init})$$

where

- Loc_1 and Loc_2 contain valuations of local variables of the first and second threads, we call them the *local stores* of the first and second thread, respectively;
- Glob contains valuations of shared variables, we call it the *global store*;
- the elements of $\mathrm{States} = \mathrm{Glob} \times \mathrm{Loc}_1 \times \mathrm{Loc}_2$ are called *program states*, the elements of $Q_1 = \mathrm{Glob} \times \mathrm{Loc}_1$ and $Q_2 = \mathrm{Glob} \times \mathrm{Loc}_2$ are called *thread states*;
- the relation $\rightarrow_1$ (resp. $\rightarrow_2$) is a binary transition relation on the states of the first (resp. second) thread;
- $\mathrm{init} \subseteq \mathrm{States}$ 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 either a step of the first or a step of the second thread. Now we define the successor operation that maps a set of program states to the set of their successors:

$$\begin{aligned}
&\mathrm{post} : 2^{\mathrm{States}} \rightarrow 2^{\mathrm{States}} \\
&\quad S \mapsto \{(g', l_1', l_2') \mid \exists\, (g, l_1, l_2) \in S : \\
&\qquad\qquad\qquad (g, l_1) \rightarrow_1 (g', l_1') \text{ and } l_2 = l_2' \\
&\qquad\qquad\quad \text{or } (g, l_2) \rightarrow_2 (g', l_2') \text{ and } l_1 = l_1'\}\,.
\end{aligned}$$

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:

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

2.2 Flanagan-Qadeer Algorithm

The algorithm of Flanagan and Qadeer from [5] tests whether a given bad state f is reachable from an initial state. The test says "no" or "don't know".
The algorithm computes sets $\mathcal{R}_i \subseteq \mathrm{Glob} \times \mathrm{Loc}_i$ and $\mathcal{G}_i \subseteq \mathrm{Glob} \times \mathrm{Glob}$ $(i = 1, 2)$ defined by the least fixed point of the following inference rules:

$$\text{INIT}\ \frac{}{\mathrm{init}_i \in \mathcal{R}_i} \qquad\qquad \text{STEP}\ \frac{(g, l) \in \mathcal{R}_i \qquad (g, l) \rightarrow_i (g', l')}{(g', l') \in \mathcal{R}_i \qquad (g, g') \in \mathcal{G}_i}$$

$$\text{ENV}\ \frac{(g, l) \in \mathcal{R}_i \qquad (g, g') \in \mathcal{G}_j}{(g', l) \in \mathcal{R}_i}\ i \neq j$$

Here, $\mathrm{init}_1 = \{(g, l_1) \mid (g, l_1, \_) \in \mathrm{init}\}$, similarly $\mathrm{init}_2 = \{(g, l_2) \mid (g, \_, l_2) \in$

init}. (The underscore means “anything”, i.e. an existentially quantified variable. The quantification is innermost, so in a formula, two underscores at different places denote different existentially quantified variables.) If $f \in \{(g, l_1, l_2) \mid (g, l_1) \in \mathcal{R}_1 \text{ and } (g, l_2) \in \mathcal{R}_2\}$, the algorithm says ”don’t know”, otherwise it says “no”.
The rules work as follows. The STEP rule discovers successors of a state of a thread that result due to a step of the same thread. Further, it stores the information about how the step changed the globals in the sets $\mathcal{G}_i$. The ENV rule uses this information to discover successors of a state of a thread that result due to communication between threads via globals. After the fixed point is reached, the set $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) contains those states of the first (resp. second) thread that the algorithm discovers. The discovered thread states contain those thread states that occur in computations.

3 Represented program states

The inference rules of the Flanagan-Qadeer algorithm define the sets $\mathcal{R}_1, \mathcal{R}_2$ of “discovered” thread states. These sets represent those program states, whose globals and locals of the first thread are in $\mathcal{R}_1$ and globals and locals of the second thread are in $\mathcal{R}_2$:

$$\{(g, l_1, l_2) \mid (g, l_1) \in \mathcal{R}_1 \text{ and } (g, l_2) \in \mathcal{R}_2\}\,.$$

Here is a small example. The program below has one global variable g that can take values 0 or 1, the first (resp. second) thread has a single local variable pc_1 (resp. pc_2), representing the program counter.

Initially $g = 0$

$A:\ \ g := 0;$	$C:\ \ g := 1;$
$B:$	$D:$

The algorithm discovers the following thread states:

$$\mathcal{R}_1 = \{(0, A), (0, B), (1, A), (1, B)\}$$

$$\mathcal{R}_2 = \{(0, C), (0, D), (1, D)\}$$

where the pair $(0, A)$, for instance, is a shorthand for the pair of two maps $([g \mapsto 0], [pc_1 \mapsto A])$. These two sets represent the set of program states

$$\{(g, l_1, l_2) \mid (g, l_1) \in \mathcal{R}_1 \text{ and } (g, l_2) \in \mathcal{R}_2\} =$$

$$\{(0, A, C), (0, A, D), (0, B, C), (0, B, D), (1, A, D), (1, B, D)\},$$

where the triple $(0, A, C)$, for instance, is a shorthand for the triple of maps $([g \mapsto 0], [pc_1 \mapsto A], [pc_2 \mapsto C])$.

4 Cartesian Abstract Interpretation

In order to characterize the Flanagan-Qadeer algorithm in the abstract interpretation framework, we first need a concrete domain, an abstract domain and a Galois connection between them:

$D = 2^{\text{States}}$ is the set underlying the concrete lattice,
$D^{\#} = 2^{Q_1} \times 2^{Q_2}$ is the set underlying the abstract lattice,
$\alpha_{\text{cart}} : D \to D^{\#}, \quad S \mapsto (T_1, T_2)$ where
$$T_1 = \{(g, l) \mid (g, l, \_) \in S\}$$
$$T_2 = \{(g, l) \mid (g, \_, l) \in S\},$$
$\gamma_{\text{cart}} : D^{\#} \to D,$
$$(T_1, T_2) \mapsto \{(g, l_1, l_2) \mid (g, l_1) \in T_1 \text{ and } (g, l_2) \in T_2\}.$$
The ordering on the concrete lattice D is inclusion, the least upper bound is the union $\cup$, the greatest lower bound is the intersection $\cap$.
The ordering on the abstract lattice $D^{\#}$ is the product ordering, i.e. $(T_1, T_2) \sqsubseteq (T_1', T_2')$ if and only if $T_1 \subseteq T_1'$ and $T_2 \subseteq T_2'$. The least upper bound $\sqcup$ is componentwise union, the greatest lower bound $\sqcap$ is componentwise intersection.

Remark that if a pair of sets (T_1, T_2) is in the image of α_{cart}, then if some global part $g \in \text{Glob}$ occurs in a thread state of one set (say, T_1), then g also occurs in a thread state of another set (i.e. T_2). So the image of the abstraction map α_{cart} is always contained in

$$D^{\#+} = \{(T_1, T_2) \in D^{\#} \mid \forall\, g \in \text{Glob} : (g, \_) \in T_1 \Leftrightarrow (g, \_) \in T_2\}\,.$$

Now we show that for the finite-state case both the cardinality and the maximal chain length of the abstract domain are in general smaller than that of the concrete domain.

Proposition 1. *Let* $\text{Glob}, \text{Loc}_1, \text{Loc}_2$ *be finite. Let* $G := |\text{Glob}| \geq 1$ *be the cardinality of the global store and* $L_1 := |\text{Loc}_1|$*,* $L_2 := |\text{Loc}_2|$ *be the cardinalities of the local stores, both at least 2 and* $l := \min\{L_1, L_2\}$*,* $L := \max\{L_1, L_2\}$*. Then the size and the maximal chain length of abstract domain are smaller and also asymptotically smaller than the size and the maximal chain length of the concrete domain. Formally:*
a) $|D^{\#}| \leq |D|$ *;*
b) $\lim_{L \to \infty} \frac{|D^{\#}|}{|D|} = 0$ *;*
c) (maximal chain length of $D^{\#}$*)* $\leq$ *(maximal chain length of* D*);*
d) $\lim_{l \to \infty} \frac{\textit{maximal chain length of } D^{\#}}{\textit{maximal chain length of } D} = 0\,.$

Proof. a)

$$|D^{\#}| = \left|2^{\text{Glob} \times \text{Loc}_1}\right| \left|2^{\text{Glob} \times \text{Loc}_2}\right| = 2^{GL_1} 2^{GL_2} =$$
$$2^{G(L_1 + L_2)} \leq 2^{GL_1 L_2} = 2^{|\text{States}|} = |D|\,.$$

b)

$$\frac{|D^{\#}|}{|D|} = \frac{\left|2^{\text{Glob}\times\text{Loc}_1}\right| \left|2^{\text{Glob}\times\text{Loc}_2}\right|}{2^{|\text{States}|}} = \frac{2^{GL_1} 2^{GL_2}}{2^{GL_1L_2}} =$$
$$= (2^G)^{L_1+L_2-L_1L_2} = (2^{-G})^{(L_1-1)(L_2-1)-1} \leq$$
$$\leq (2^{-G})^{(L-1)(2-1)-1} = 2^{-G(L-2)} \xrightarrow{L\to\infty} 0\,.$$

c) A longest chain in the concrete domain $D = 2^{\text{Glob}\times\text{Loc}_1\times\text{Loc}_2}$ has $1 + |\text{Glob} \times \text{Loc}_1 \times \text{Loc}_2| = 1 + GL_1L_2$ elements, its minimum is the empty set $\emptyset$ and its maximum is States. Here is a longest chain in the abstract domain:

$$(\emptyset, \emptyset) \sqsubset \ldots \sqsubset (Q_1, \emptyset) \sqsubset \ldots \sqsubset (Q_1, Q_2)\,,$$

where first the elements of Q_1 are added to the first component in some order one by one and then the elements of Q_2. This chain has $1 + |Q_1| + |Q_2| = 1 + G(L_1 + L_2)$ elements, which is smaller than the maximal chain length in the concrete domain. Assume some other longest chain in the concrete domain is given. Then for each two adjacent elements $(A_1, A_2) \sqsubset (B_1, B_2)$ of the chain we have that either $A_1 = B_1$ and $A_2 \mathbin{\dot\cup} \{\_\} = B_2$ or $A_2 = B_2$ and $A_1 \mathbin{\dot\cup} \{\_\} = B_1$ (otherwise the chain could be made longer). So one can construct the chain by starting with $(\emptyset, \emptyset)$ and adding elements one by one in some order to the first or to the second component. The number of such additions is bounded by the sizes of the components, namely, by $|Q_1|$ and $|Q_2|$. So totally $|Q_1| + |Q_2|$ additions can be performed. The number of elements in the chain is thus also $|Q_1| + |Q_2| + 1$.

d)

$$\lim_{l\to\infty} \frac{\text{maximal chain length of } D^{\#}}{\text{maximal chain length of } D} = \lim_{l\to\infty} \frac{1 + G(L_1 + L_2)}{1 + GL_1L_2} =$$
$$\underbrace{\lim_{l\to\infty} \frac{1}{1 + GL_1L_2}}_{0} + \lim_{l\to\infty} \frac{G(L_1 + L_2)}{1 + GL_1L_2} = \lim_{l\to\infty} \frac{1}{\frac{1}{GL_1} + L_2} + \lim_{l\to\infty} \frac{1}{\frac{1}{GL_2} + L_1} = 0\,.$$

□

Two remarks should be made. First, if only one local store grows but the other remains constant-size, then the quotient $\frac{\text{maximal chain length of } D^{\#}}{\text{maximal chain length of } D}$ approaches some small positive value between 0 and 1. In case the number of threads is not two, but variable (say, n), we get similar asymptotic results for $n \to \infty$.

From now on, we sometimes omit the parentheses around the argument of a map, writing, e.g. fx for $f(x)$.

Proposition 2. *The pair of maps* $(\alpha_{\text{cart}}, \gamma_{\text{cart}})$ *is a Galois connection, i.e. for all* $S \in D, (T_1, T_2) \in D^{\#}$ *holds*

$$\alpha_{\text{cart}} S \sqsubseteq (T_1, T_2) \textit{ iff } S \subseteq \gamma_{\text{cart}}(T_1, T_2)\,.$$

Proof. "⇒": Let $(g, l_1, l_2) \in S$. Let $(T_1', T_2') = \alpha_{\text{cart}} S$. Then by definition of α_{cart} we have $(g, l_1) \in T_1' \subseteq T_1$ and $(g, l_2) \in T_2' \subseteq T_2$. So $(g, l_1, l_2) \in \gamma_{\text{cart}}(T_1, T_2)$ by definition of γ_{cart}.
"⇐": Let $(T_1', T_2') = \alpha_{\text{cart}} S$. Let $(g, l_1) \in T_1'$. By definition of α_{cart} there is an l_2 with $(g, l_1, l_2) \in S \subseteq \gamma_{\text{cart}}(T_1, T_2)$. By definition of γ_{cart} we have $(g, l_1) \in T_1$. So $T_1' \subseteq T_1$. Analogously we get $T_2' \subseteq T_2$. □

Other definitions of Galois connection in the literature require that the abstraction and concretization maps are monotonic. However, this follows from our definition of Galois connection; the proof is left as an exercise for the reader.

5 Flanagan-Qadeer Algorithm as Cartesian Abstract Fixpoint Checking

5.1 Theorem and Example

We know (e.g. from [2]) that if the abstraction and concretization maps are given by a Galois connection (α, γ) between an abstract and a concrete domain, then the abstraction of the program is usually defined as the least fixed point of $\lambda T.\alpha(\text{init} \cup \text{post}\gamma T)$. Recall that the Flanagan-Qadeer algorithm computes $\mathcal{R}_1$ and $\mathcal{R}_2$, the sets of "discovered" states of the first and second thread.

Theorem 3. *[Thread-Modular Model Checking is Cartesian Abstract Interpretation]*
The output of the Flanagan-Qadeer algorithm is the least fixed point of the abstract fixpoint checking operator with the abstraction map α_{cart} and concretization map γ_{cart}. Formally:

$$(\mathcal{R}_1, \mathcal{R}_2) \;=\; \text{lfp}\ \lambda T.\ \alpha_{\text{cart}}(\text{init} \cup \text{post}\gamma_{\text{cart}} T)\,.$$

It is not clear why this is so and how the assumptions are connected. For our tiny example, the right hand of the above equation (i.e. the least fixed point) is

$$(\{(0, A), (0, B), (1, A), (1, B)\},\ \ \{(0, C), (0, D), (1, D)\})\ ,$$

which coincides with $(\mathcal{R}_1, \mathcal{R}_2)$ computed by the algorithm. We prove that the left and right hand side always coincide in the next section.

5.2 Proof

First we transform the inference rules of the Flanagan-Qadeer algorithm by getting rid of the sets $\mathcal{G}_1$ and $\mathcal{G}_2$. We get an equivalent system of inference rules

$$\text{INIT}_1' \ \frac{}{\text{init}_1 \in \mathcal{R}_1} \qquad \text{STEP}_1' \ \frac{(g, l) \in \mathcal{R}_1 \qquad (g, l) \to_1 (g', l')}{(g', l') \in \mathcal{R}_1}$$

$$\text{ENV}_1' \ \frac{(g, l) \in \mathcal{R}_1 \qquad (g, l_2) \in \mathcal{R}_2 \qquad (g, l_2) \to_2 (g', \_)}{(g', l) \in \mathcal{R}_1}$$

The rules INIT_2', STEP_2' and ENV_2' are analogous to INIT_1', STEP_1' and ENV_1' where the indices 1 and 2 are exchanged. Remark that init_1 and init_2 contain thread states with the same global parts. Also remark that whenever $(g, l) \in \mathcal{R}_1$ and $(g, l) \rightarrow_1 (g', \_)$ and there is some thread state $(g, \_)$ in $\mathcal{R}_2$, then both rules STEP_1' and ENV_2' apply, giving two thread states for $\mathcal{R}_1$ and $\mathcal{R}_2$ with the same global part g'. Similarly, whenever $(g, l) \in \mathcal{R}_2$ and $(g, l) \rightarrow_2 (g', \_)$ and there is some thread state $(g, \_)$ in $\mathcal{R}_1$, then both rules STEP_2' and ENV_1' apply, giving two thread states for $\mathcal{R}_2$ and $\mathcal{R}_1$ with the same global part g'. By induction follows that whenever there is a thread state in $\mathcal{R}_i$ with some global part g, there is a thread state in $\mathcal{R}_j$ with the same global part g ($i \neq j$).
This means that we can combine the STEP' and ENV' rules into one rule. The following system of inference rules defines the same sets $\mathcal{R}_1, \mathcal{R}_2$ as the system above:

$$\mathrm{INIT}_1' \; \frac{}{\mathrm{init}_1 \in \mathcal{R}_1} \qquad \mathrm{INIT}_2' \; \frac{}{\mathrm{init}_2 \in \mathcal{R}_2}$$

$$\mathrm{POST}_1^{\#} \; \frac{(g, l_1) \in \mathcal{R}_1 \qquad (g, l_2) \in \mathcal{R}_2 \qquad (g, l_2) \rightarrow_2 (g', l_2')}{(g', l_1) \in \mathcal{R}_1 \qquad (g', l_2') \in \mathcal{R}_2}$$

$$\mathrm{POST}_2^{\#} \; \frac{(g, l_2) \in \mathcal{R}_2 \qquad (g, l_1) \in \mathcal{R}_1 \qquad (g, l_1) \rightarrow_1 (g', l_1')}{(g', l_2) \in \mathcal{R}_2 \qquad (g', l_1') \in \mathcal{R}_1}$$

Each $\mathrm{POST}^{\#}$ rule takes two sets (called $\mathcal{R}_1$ and $\mathcal{R}_2$ above) and gives new elements for the first and new elements for the second set. All possible applications of the $\mathrm{POST}^{\#}$ rules on any fixed pair of sets (T_1, T_2) can be expressed as computing

$$\begin{aligned} p^{\#}(T_1, T_2) = \{((g', l_1'), (g', l_2')) \mid \exists\, ((g, l_1), (g, l_2)) \in T_1 \times T_2 :& \\ (g, l_1) \rightarrow_1 (g', l_1') \text{ and } l_2 = l_2'& \\ \text{or } (g, l_2) \rightarrow_2 (g', l_2') \text{ and } l_1 = l_1'\},& \end{aligned}$$

the new elements being the first and second projection of the result. Thus, applying the $\mathrm{POST}^{\#}$ rules corresponds to computing

$$\mathrm{post}^{\#}(T_1, T_2) := (\pi_1 p^{\#}(T_1, T_2), \pi_2 p^{\#}(T_1, T_2))\,,$$

i.e. applying the map

$$\lambda(T_1, T_2)\,.\,\mathrm{post}^{\#}(T_1, T_2)\,.$$

Remark that $(\mathrm{init}_1, \mathrm{init}_2) = \alpha_{\mathrm{cart}}\mathrm{init}$. Then the pair of computed sets $(\mathcal{R}_1, \mathcal{R}_2)$ is the least fixed point of

$$\lambda T\,.\,\alpha_{\mathrm{cart}}\mathrm{init} \sqcup \mathrm{post}^{\#} T\,.$$

It is interesting to see that $\mathrm{post}^{\#}$ can be expressed in terms of post and the abstraction/concretization maps:

Proposition 4. *For any $T \in D^{\#}$ holds:*

$$\mathrm{post}^{\#} T = \alpha_{\mathrm{cart}} \mathrm{post}\, \gamma_{\mathrm{cart}} T\,.$$

Proof. Let $(T_1, T_2) := T$.
"$\sqsubseteq$":
Let $(g', l_1') \in \pi_1 p^\# T$. Then there is an l_2' so that the pair $((g', l_1'), (g', l_2')) \in p^\# T$. Then there are g, l_1, l_2 with $(g, l_1) \in T_1$, $(g, l_2) \in T_2$ and

$$\begin{aligned} &(g, l_1) \rightarrow_1 (g', l_1') \text{ and } l_2 = l_2' \\ \text{or } &(g, l_2) \rightarrow_2 (g', l_2') \text{ and } l_1 = l_1' \,. \end{aligned}$$

Then $(g, l_1, l_2) \in \gamma_{\mathrm{cart}}(T_1, T_2) = \gamma_{\mathrm{cart}} T$ by definition of γ_{cart}. So $(g', l_1', l_2') \in \mathrm{post}\gamma_{\mathrm{cart}} T$ by definition of the successor map post and thus (g', l_1') is in the first component of $\alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T$. So $\pi_1 p^\# T$ is included in the first component of $\alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T$.
Analogously follows that $\pi_2 p^\# T$ is included in the second component of $\alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T$.
"$\sqsupseteq$":
Let (g', l_1') be in the first component of $\alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T$. Then there is an l_2' with $(g', l_1', l_2') \in \mathrm{post}\gamma_{\mathrm{cart}} T$. So there are g, l_1, l_2 with $(g, l_1, l_2) \in \gamma_{\mathrm{cart}} T$ and

$$\begin{aligned} &(g, l_1) \rightarrow_1 (g', l_1') \text{ and } l_2 = l_2' \\ \text{or } &(g, l_2) \rightarrow_2 (g', l_2') \text{ and } l_1 = l_1' \,. \end{aligned}$$

From $(g, l_1, l_2) \in \gamma_{\mathrm{cart}} T$ we know that $(g, l_1) \in T_1$ and $(g, l_2) \in T_2$. By definition of $p^\#$ we have $((g', l_1'), (g', l_2')) \in p^\#(T_1, T_2)$. So $(g', l_1') \in \pi_1 p^\# T$. We have shown that the first component of $\alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T$ is included in $\pi_1 p^\# T$, which is the first component of $\mathrm{post}^\# T$.
Analogously one can show that the second component of $\alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T$ is included in $\pi_2 p^\# T$, which is the second component of $\mathrm{post}^\# T$.

□

So the algorithm computes the least fixed point of

$$\begin{aligned} &\lambda T \,.\, \alpha_{\mathrm{cart}}\mathrm{init} \sqcup \alpha_{\mathrm{cart}}\mathrm{post}\gamma_{\mathrm{cart}} T \\ &= \lambda T \,.\, \alpha_{\mathrm{cart}}(\mathrm{init} \cup \mathrm{post}\gamma_{\mathrm{cart}} T) \,. \end{aligned}$$

So

$$(\mathcal{R}_1, \mathcal{R}_2) \;=\; \mathrm{lfp}\ \lambda T.\ \alpha_{\mathrm{cart}}(\mathrm{init} \cup \mathrm{post}\gamma_{\mathrm{cart}} T) \,.$$

6 Local Cartesian Abstraction

Up to now we identified the Flanagan-Qadeer model-checking as abstract fixpoint checking on an abstract domain. However, it turns out that the output of the Flanagan-Qadeer algorithm can also be characterized by a very simple abstraction of multithreaded programs which is defined on the concrete domain. Now we define this abstraction.
First recall that the *Cartesian abstraction of a set of pairs* is the smallest Cartesian product containing this subset. We define it formally as

$$\begin{aligned} &\mathcal{C}^\# : 2^{X_1 \times X_2} \rightarrow 2^{X_1 \times X_2}, \\ &P \mapsto \{(s_1, s_2) \mid (s_1, \_) \in P \text{ and } (\_, s_2) \in P\} \,, \end{aligned}$$

where X_1 and X_2 are any sets (e.g. Q_1 and Q_2). We have $\mathcal{C}^{\#}P = \pi_1 P \times \pi_2 P$. An analog of Cartesian abstraction on the concrete domain $D = 2^{\text{States}}$ is

$$\begin{aligned} \mathcal{C} : D &\to D \\ S &\mapsto \{(g, l_1, l_2) \mid (g, l_1, \_) \in S \text{ and } (g, \_, l_2) \in S\}\,. \end{aligned}$$

We call this map *local Cartesian abstraction* of a set of program states since it simplifies to the Cartesian abstraction of a set of pairs if Glob is a singleton.
It turns out that local Cartesian abstraction is representable in the abstract interpretation framework.

Proposition 5. *Local Cartesian abstraction is approximation with the abstraction map α_{cart} and the concretization map γ_{cart}. Formally:*

$$\mathcal{C} = \gamma_{\text{cart}} \alpha_{\text{cart}}$$

Proof. Let $S \subseteq$ States. We have to show $\mathcal{C}S = \gamma_{\text{cart}}\alpha_{\text{cart}}S$. Let $(T_1, T_2) = \alpha_{\text{cart}}S$. We have
$(g, l_1, l_2) \in \mathcal{C}S$ iff [definition of $\mathcal{C}$]
$(g, l_1, \_) \in S$ and $(g, \_, l_2) \in S$ iff [definition of α_{cart}]
$(g, l_1) \in T_1$ and $(g, l_2) \in T_2$ iff [definition of γ_{cart}]
$(g, l_1, l_2) \in \gamma_{\text{cart}}(T_1, T_2) = \gamma_{\text{cart}}\alpha_{\text{cart}}S$. □

7 Thread-Modular Model-Checking as Local Cartesian Abstraction

7.1 Theorem and Example

Given an abstraction map α and a concretization map γ between an abstract and a concrete domain, we can alternatively perform the abstract fixed point checking in the concrete domain (in contradiction to the computation in the abstract domain as before). Then the least fixed point of $\lambda S.\gamma\alpha(\text{init} \cup \text{post}S)$ is computed.
For our special Galois connection $(\alpha_{\text{cart}}, \gamma_{\text{cart}})$ this means the following. We can enumerate the states of the program by iterative successor computation and at each step overapproximate by local Cartesian abstraction. Naively implemented, this algorithm would require exponential time (in number of threads, if it is not constant). However, it turns out that the Flanagan-Qadeer algorithm solves the same problem in polynomial time.

Theorem 6. *[Thread-Modular Model-Checking as Local Cartesian abstraction] The concretization of the output of the Flanagan-Qadeer algorithm is equal to the result of abstract fixpoint checking with local Cartesian abstraction. Formally:*

$$\gamma(\mathcal{R}_1, \mathcal{R}_2) \;=\; \text{lfp}\ \lambda S.\ \mathcal{C}(\text{init} \cup \text{post}S). \tag{1}$$

For our tiny example, let us compute the least fixed point of $\lambda S.\mathcal{C}(\text{init} \cup \text{post} S)$ by definition. The corresponding chain is

$$\begin{aligned} &\{(0, A, C)\} \\ \sqsubset\ &\{(0, A, C), (0, B, C), (1, A, D)\} \\ \sqsubset\ &\{(0, A, C), (0, B, C), (1, A, D), (1, B, D), (0, B, D), (0, A, D)\} \quad \text{(fixed point)}\,, \end{aligned}$$

the last term being the right hand side of (1). The left and the right hand side coincide in this example. Now we prove that they always coincide.

7.2 Proof Preparations

Before we start proving the theorem, let us prove a basic fact about the abstract fixpoint checking.
Let D be a complete lattice with ordering $\subseteq$, bottom element $\emptyset$, join $\cup$, meet $\cap$ (concrete lattice). Further, let $D^\#$ be a complete lattice with ordering $\sqsubseteq$, bottom element $\perp$, join $\sqcup$ and meet $\sqcap$ (abstract lattice). Let a pair of maps $\alpha : D \to D^\#$ and $\gamma : D^\# \to D$ be a Galois connection between the concrete and abstract lattices, i.e. for all concrete elements $x \in D$ and all abstract elements $y \in D^\#$ we have $\alpha x \subseteq y$ iff $x \sqsubseteq \gamma y$. Let $F : D \to D$ be any monotone map and $\text{init} \in D$ any concrete element. Further, we call $\rho := \gamma\alpha : D \to D$ the *overapproximation* operator. (Compare these standard definitions to the original Cousots' work [4].)
One way to perform abstract fixpoint checking is to compute the least fixed point of $G^\# = \lambda T.\ \alpha(\text{init} \cup F\gamma T)$ in the abstract lattice. The other way is to compute the least fixed point of $G = \lambda S.\ \gamma\alpha(\text{init} \cup FS)$ in the concrete lattice.
One would expect that these two fixed points are the same up to abstraction/concretization. Now we show that this is indeed the case under a special assumption about the concretization map γ.

Hypothesis: The concretization map γ is semi-continuous, i.e. for all ascending chains $X \subseteq D^\#$ we have $\gamma(\sqcup X) = \cup\gamma X$.
This hypothesis is especially satisfied for a continuous γ, i.e. when for all chains $X \subseteq D^\#$ we have $\gamma(\sqcup X) = \cup\gamma X$.

Let μ be any ordinal that is strictly greater than the cardinalities of D and $D^\#$. Let us define two sequences with indices from μ:

For $k = 0$:	$T^0 = \alpha\text{init}$	$S^0 = \rho\,\text{init}\,,$
for successor ordinals $k + 1 \in \mu$:	$T^{k+1} = G^\# T^k$	$S^{k+1} = GS^k\,,$
for limit ordinals $k \in \mu$:	$T^k = \bigsqcup_{k'<k} T^{k'}$	$S^k = \bigcup_{k'<k} S^{k'}\,.$

From [3] Cor. 3.3 we know that the sequences $(S^k)_k$, $(T^k)_k$ are stationary increasing chains and the limits are the least fixed points over αinit and $\rho\,\text{init}$, respectively. One can show that if we start the sequences $(T^k)_k$ and $(S^k)_k$ with the bottom elements $\perp$ and $\emptyset$ (instead of αinit and $\rho\,\text{init}$), the limits would be

the same, so the limits are the least fixed points (over $\perp$ and $\emptyset$).
The hypothesis immediately implies that for each limit ordinal $k \in \mu$ holds

$$\gamma \bigsqcup_{k'<k} T^{k'} = \bigcup_{k'<k} \gamma T^{k'} \quad \text{and} \quad \gamma \bigsqcup_{k'<k} \alpha S^{k'} = \bigcup_{k'<k} \gamma\alpha S^{k'} . \tag{2}$$

Using this formula we show step by step that the least fixed points of G and $G^{\#}$ coincide up to abstraction and concretization.
We need some basic facts about Galois connections and about the overapproximation operator (see e.g. [1], [4]). From the definition of Galois connection one can prove that the abstraction and concretization maps α and γ are monotone. Further overapproximation map ρ is idempotent. Further, overapproximating and then abstracting is the same as abstracting: $\alpha\rho = \alpha$ (since $\gamma\alpha$ is extensive, $\alpha\gamma$ reductive and both are monotone).
First we show that overapproximating S^k doesn't change it.

Proposition 7. *Each S^k is invariant under overapproximation. Formally:*

$$\forall\, k \in \mu \;:\quad \rho S^k = S^k .$$

Proof. We use transfinite induction.
For $k = 0$, we have $\rho S^0 = \rho\rho\,\mathrm{init} \overset{\rho \text{ idempotent}}{=} \rho\,\mathrm{init} = S^0$.
For a successor ordinal $k + 1$, we have $\rho S^{k+1} = \rho\rho(\mathrm{init} \cup FS^k) \overset{\rho \text{ idempotent}}{=} \rho(\mathrm{init} \cup FS^k) = S^{k+1}$.
If k is a limit ordinal, then $\rho S^k = \gamma\alpha \bigcup_{k'<k} S^{k'} \overset{\alpha \text{ complete join morphism}}{=} \gamma \bigsqcup_{k'<k} \alpha S^{k'} \overset{\text{formula(2)}}{=} \bigcup_{k'<k} \gamma\alpha S^{k'} \overset{\rho=\gamma\alpha}{=} \bigcup_{k'<k} \rho S^{k'} \overset{\text{induction assumption}}{=} \bigcup_{k'<k} S^{k'} = S^k$. □

Proposition 8. *Each T^k is the abstraction of S^k. Formally:*

$$\forall\, k \in \mu \;:\quad T^k = \alpha S^k .$$

Proof. Transfinite induction.
For $k = 0$ we have $T^0 = \alpha\mathrm{init} \overset{\alpha=\alpha\rho}{=} \alpha\rho\mathrm{init} = \alpha S^0$.
For a successor ordinal $k + 1$ we have $T^{k+1} = \alpha(\mathrm{init} \cup F\gamma T^k) \overset{\text{induction assumption}}{=} \alpha(\mathrm{init} \cup F\gamma\alpha S^k) \overset{\gamma\alpha=\rho}{=} \alpha(\mathrm{init} \cup F\rho S^k) \overset{\rho S^k=S^k}{=} \alpha(\mathrm{init} \cup FS^k) \overset{\alpha=\alpha\rho}{=} \alpha\rho(\mathrm{init} \cup FS^k) = \alpha S^{k+1}$.
If k is a limit ordinal, then $T^k = \bigsqcup_{k'<k} T^{k'} \overset{\text{induction assumption}}{=} \bigsqcup_{k'<k} \alpha S^{k'} \overset{\alpha \text{ complete join morphism}}{=} \alpha \bigcup_{k'<k} S^{k'} = \alpha S^k$. □

Proposition 9. *Each S^k is the concretization of T^k. Formally:*

$$\forall\, k \in \mu \;:\quad \gamma T^k = S^k .$$

Proof. Transfinite induction.
For $k = 0$ we have $\gamma T^0 = \gamma\alpha\mathrm{init} = \rho\mathrm{init} = S^0$.

For a successor ordinal $k+1$ we have $\gamma T^{k+1} = \gamma\alpha(\mathrm{init} \cup F\gamma T^k) = \rho(\mathrm{init} \cup F\gamma T^k) \overset{\text{induction assumption}}{=} \rho(\mathrm{init} \cup FS^k) = S^{k+1}$.
If k is a limit ordinal, $\gamma T^k = \gamma \bigsqcup_{k'<k} T^{k'} \overset{\text{formula(2)}}{=} \bigcup_{k'<k} \gamma T^{k'} \overset{\text{induction assumption}}{=} \bigcup_{k'<k} S^{k'} = S^k$.

□

Let $\lambda \in \mu$ be any ordinal at which both sequences are stationary, i.e. $S^\lambda = S^{\lambda+1}$ and $T^\lambda = T^{\lambda+1}$. Then the least fixed point of G is S^λ and the least fixed point of $G^\#$ is T^λ. Propositions 8 and 9 imply the following

Theorem 10. *The least fixed points of G and $G^\#$ coincide up to abstraction and concretization:*

$$\gamma \text{ lfp } G^\# = \text{lfp } G \;\; \textit{and} \;\; \text{lfp } G^\# = \alpha \text{ lfp } G\,.$$

7.3 Proof conclusion and remarks

We now show that for our Galois connection $(\alpha_{\mathrm{cart}}, \gamma_{\mathrm{cart}})$ between our domains D and $D^\#$ the hypothesis holds.

Proposition 11. γ_{cart} *is continuous, i.e. for all chains $X \subseteq D^\#$ holds:*

$$\gamma_{\mathrm{cart}}(\sqcup X) \;=\; \cup\, \gamma_{\mathrm{cart}} X\,.$$

Proof. "$\subseteq$". Let $(g, l_1, l_2) \in \gamma_{\mathrm{cart}}(\sqcup X)$. Then (g, l_1) (resp. (g, l_2)) is in the first (resp. second) component of $\sqcup X$. Then there are (T_1, T_2) and (T_1', T_2') in X with $(g, l_1) \in T_1$ and $(g, l_2) \in T_2'$. Since X is a chain, we have either $(T_1, T_2) \sqsupseteq (T_1', T_2')$ or $(T_1, T_2) \sqsubseteq (T_1', T_2')$. Without loss of generality let $(T_1, T_2) \sqsupseteq (T_1', T_2')$. Then $(g, l_2) \in T_2$, so $(g, l_1, l_2) \in \gamma_{\mathrm{cart}}(T_1, T_2) \subseteq \cup\gamma_{\mathrm{cart}} X$.
"$\supseteq$" holds by monotonicity of γ_{cart} and definition of the least upper bound. □

The map post : $D \to D$ is monotone. Proposition 5 and Theorems 3 and 10 imply

$$\gamma(\mathcal{R}_1, \mathcal{R}_2) \;=\; \text{lfp } \lambda S.\, \mathcal{C}(\mathrm{init} \cup \mathrm{post} S)\,,$$

which concludes the proof of Theorem 6.

The whole proof is very general and would get shorter for the finite-state case. The current proof depends only on the fact that the concretization map is semi-continuous. It depends neither on the exact structure of the abstract and the concrete lattices, nor on the exact definition of the abstraction/concretization maps, nor on the structure of the monotone map F. In applications of the abstract interpretation framework, the concretization map is often semi-continuous, so that Theorem 10 can be reused.

8 Precision vs. Runtime

For speaking about runtime, assume that all the domains are finite. The definitions of the abstract and concrete domains as well as the corresponding Galois connection and the resulting local Cartesian abstraction extend to n threads in the natural way:
$\text{States} = \text{Glob} \times \prod_{i=1}^{n} \text{Loc}_i$,
$D = 2^{\text{States}}$,
$D^{\#} = \prod_{i=1}^{n} \text{Glob} \times \text{Loc}_\text{i}$,
$(T_1, ..., T_n) \sqsubseteq (T_1', ..., T_n')$ iff for all $1 \leq i \leq n$ holds $T_i \subseteq T_i'$,

$$\alpha_{\text{cart}} : D \to D^{\#}, \quad S \mapsto (T_1, ...T_n) \text{ where}$$
$$T_i = \pi_{\text{Glob}\times\text{Loc}_\text{i}} S \quad (i = 1, ..., n),$$

$$\gamma_{\text{cart}} : D^{\#} \to D,$$
$$(T_1, ..., T_n) \mapsto \{(g, l_1, ..., l_n) \mid \forall i : (g, l_i) \in T_i\},$$

$\mathcal{C} = \gamma_{\text{cart}} \alpha_{\text{cart}}$,
$\text{post}^{\#} = \alpha_{\text{cart}} \text{post} \gamma_{\text{cart}}$,

where $\pi_{\text{Glob}\times\text{Loc}_\text{i}}$ is the projection map onto the component $\text{Glob} \times \text{Loc}_\text{i}$.

The previous results formulated for $n = 2$ also hold for all numbers of threads $n \geq 2$.

8.1 Boundary of the Flanagan and Qadeer algorithm

Now we try to push the Flanagan-Qadeer algorithm to increase precision without losing polynomial complexity.

A usual way to change a search algorithm is to use "frontier search", which forgets the nodes of the search graph that were expanded in the past. The idea is to apply local Cartesian abstraction not to all of the discovered states, but only to the latest discovered states:

$$T^0 = \alpha_{\text{cart}} \text{init} \quad \text{ and } \quad T^{i+1} = \text{post}^{\#} T^i \ (i \geq 0).$$

The sequence stops when $\gamma_{\text{cart}} T^{k+1} \subseteq \cup_{i=0}^{k} \gamma_{\text{cart}} T^i$. For this k we can show that $X := \cup_{i=0}^{k} \gamma_{\text{cart}} T^i$ is an inductive invariant, i.e. $\text{init} \subseteq X$ and $\text{post} X \subseteq X$.
We can implement this iteration in the abstract domain $D^{\#}$ as follows. Consider following inference rule ($i \neq j$):

$$\text{POST}^{\#}_{\text{ij}} \frac{(g, l_i) \in \mathcal{R}_i \qquad (g, l_j) \in \mathcal{R}_j \qquad (g, l_j) \to_j (g', l_j')}{(g', l_i) \in \mathcal{R}_i' \qquad (g', l_j') \in \mathcal{R}_j'} \ .$$

Except for the primed versions $\mathcal{R}_i'$ and $\mathcal{R}_j'$ in the conclusion of the rule, this is the same rule that is used in the reformulation of the Flanagan-Qadeer algorithm. If $T^i = (\mathcal{R}_1, ..., \mathcal{R}_n)$ is the ith element of the iteration sequence then $T^{i+1} = (\mathcal{R}_1', ..., \mathcal{R}_n')$ is computed by the above rule. So each step of the new iteration scheme is polynomial. It turns out that frontier search breaks the polynomial-time boundary of the method:

Theorem 12. *Frontier search with Cartesian abstraction has exponential worst-case runtime in the number of threads.*

Proof. It suffices to give an example program on which the new algorithm needs exponential time in the number of threads n. However, even the Flanagan-Qadeer may have exponential runtime in n if the size of the global store is exponential in n. So we require that the size of the description of the program (the global store, the local stores, the transition relations of the threads and the initial state set) is polynomial in n. It suffices to present a family of multithreaded programs so that:

1. the nth program in the family has n threads;
2. the sizes of the global store and local stores are polynomial in n;
3. each program of the family has exactly one run which is exponentially long in n.

Assume such a program with a single initial state is given. If for some $i \geq 0$ the components of the tuple T^i contain at most one element each, then $\gamma_{\text{cart}} T^i$ contains at most one element, and hence $\text{post}\gamma_{\text{cart}} T^i$ has at most one element, so $T^{i+1} = \alpha_{\text{cart}}\text{post}\gamma_{\text{cart}} T^i$ is a tuple of sets which are either all empty sets or all singletons. Since $\gamma_{\text{cart}} T^i$ contains $\text{post}^i(\text{init})$, we inductively follow that $\gamma_{\text{cart}} T^i = \text{post}^i(\text{init})$ for all $i \geq 0$, i.e. no approximation happens. Especially the sequence $(T^i)_{0 \leq i \leq k}$ is exponentially long.

Now we give a family of programs satisfying the conditions above.

Example 13. Below you see a program with n threads whose single run has length at least 2^n. The size of the global store is $n+1$, the local stores have two elements each. The statements in brackets $<>$ are atomic.

Global boolean variable with initial value:
$t = 1$ (takes values from $\{0, ..., n\}$)
Thread 1:
```
0: wait until t = 1;
1: <t := 2;   goto 0;>
```
Thread i $(1 < i < n)$:
```
0: <wait until t = i;   t := 1;>
1: <wait until t = i;   t := i + 1;   goto 0;>
```
Thread n:
```
0: <wait until t = n;   t := 1;>
1: <wait until t = n;   t := 0;   goto 0;>
```

The program implements a binary counter. The program performs repeated increment with the school addition method. The local store of the ith thread represents the position $i - 1$ of the number $(1 \leq i \leq n)$. The carry position is stored in the global variable t. The value $t = 0$ means the carry is nowhere.

Below is the single run for $n = 3$ where pc_i is the program counter of the ith thread. Each column represents a state of the whole program, a successor state is to the right of its predecessor:

variable	*	*		*	*			*	*		*	*			
t	1	1	2	1	1	2	3	1	1	2	1	1	2	3	0
pc_1	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0
pc_2	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0
pc_3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0

Let us consider the columns marked by the star (*), i.e the columns with $t = 1$ (the carry is above the 0th position). One sees that the values of the program counters (pc_3, pc_2, pc_1) evolve like a binary counter. The formal inductive proof that this program implements the binary counter is left as an exercise for the reader. $\square$

So frontier search is the precision limit of Cartesian Abstraction.

Remark that the set of reachable states of binary counter in Example 13 is so big that the output of the Flanagan-Qadeer algorithm is exact:

$$
\begin{aligned}
\mathcal{R}_1 &= \{(1,0),(1,1),(2,0), \quad (3,0), \quad (0,0)\}\,,\\
\mathcal{R}_2 &= \{(1,0),(1,1),(2,0),(2,1), \quad (3,0), \quad (0,0)\}\,,\\
\mathcal{R}_3 &= \{(1,0),(1,1),(2,0),(2,1), \quad (3,0),(3,1), \quad (0,0)\}\,.
\end{aligned}
$$

Namely, the concretization of the output $\gamma_{\text{cart}}(\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3)$ gives exactly the set of reachable states. So this is also an example of when an attempt of regaining precision by frontier search doesn't increase precision but requires exponential time.

The binary counter example has a property that the size of the global store grows linearly with the number of threads. Can one get states reachable only by exponentially long runs with a sublinear or even constant size of the global store? This would give an answer to the question whether exponentially long runs occur only due to the growing number of threads or need a growing global store. We pose the following

Problem 14. Prove or give a counterexample. There is no family $(P_n)_{n\geq 1}$ of multithreaded programs so that

1. the nth program P_n consists of n threads;
2. the sizes of the global and local stores are bounded by a constant (independent on n);
3. the diameter of the transition graph of the nth program P_n is asymptotically exponential in n, i.e. there is a constant $c > 1$ so that for almost all $n \in \mathbb{N}$ the diameter of the transition graph of P_n exceeds c^n.

We do not see how to solve this problem at the moment.

8.2 Regaining precision

The inherent problem of the original Flanagan-Qadeer algorithm is that its abstraction is too coarse. We show two techniques that overcome precision loss and give practical polynomial-time algorithms based on it.

There are two sources of precision loss. One is that local Cartesian abstraction "forgets" temporal properties of the program. That means that a state of one thread that occurs late in any computation can be combined with a state of another thread that occurs early in any computation.

Example 15. In this example, the Flanagan-Qadeer algorithm cannot prove that the first thread never reaches label D:
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:                 ||
```

The algorithm computes the sets

$$\begin{aligned}\mathcal{R}_1 =&\{(0,A),(0,B),(0,C),(0,D),\\&(1,A),(1,B),(1,C),(1,D)\}\,,\\ \mathcal{R}_2 =&\{(0,E),(0,G),(1,F)\}\,.\end{aligned}$$

Since $(\_, D) \in \mathcal{R}_1$, the label D is discovered.

The second source of precision loss is that local Cartesian abstraction "forgets" dependencies between threads even in successors of a single program state. For instance, the Flanagan-Qadeer algorithm is unable to prove mutual exctusion with locks (see example in [5]) of the following program.

Example 16. In the program below, execution never reaches the state $pc_1 = pc_2 = B$.
Global variable m=0

```
Thread 1:            ||Thread 2:
A: <wait until m=0;  ||A: <wait until m=0;
      m:=1;>         ||      m:=1;>
B:                   ||B:
```

The algorithm computes the sets $\mathcal{R}_1 = \mathcal{R}_2 = \{(0,A),(1,A),(1,B)\}$, whose concretization $\gamma_{cart}(\mathcal{R}_1, \mathcal{R}_2)$ contains $(1,B,B)$, which violates mutual exclusion.

Now we give two techniques that solve the above problems while still being polynomial in time.

Relaxed Frontier Search. Frontier search is a boundary of the thread-modular method. It takes exponential time in the worst case. Our new method is a combination of the original Flanagan-Qadeer algorithm and the frontier search. It has polynomial runtime.

Definition 17 (Relaxed Frontier Sequence). *Let the sequence* $(T^i)_{i\geq 0}$ *of elements of* $D^{\#}$ *be defined recursively by*

$T^0 = \alpha_{\text{cart}}\text{init},$
$T^{i+1} = \begin{cases} \text{post}^\# T^i & , \textit{ if } \text{post}^\# T^i \not\sqsubseteq \sqcup_{j=0}^{i} T^j, \\ T^i \sqcup \text{post}^\# T^i & , \textit{ if } \text{post}^\# T^i \sqsubseteq \sqcup_{j=0}^{i} T^j. \end{cases}$
Let $k = \min\{i \mid T^{i+1} \sqsubseteq T^i\}$.

This (in general non-monotone) sequence is the basis of our algorithm, which needs only elements of the sequence up to position k. First we prove that k is well-defined, and is at most quadratic in the number of threads. Let $G = |\text{Glob}|$ the size of the global store and L the maximal size of the local store, i.e. $L = \max_i |\text{Loc}_\text{i}|$.

Proposition 18. *The value* k *is well-defined and is at most* $nGL(nGL+1) = O((nGL)^2)$.

Proof. First we prove that k is well-defined. The set $D^\#$ is finite, so the monotone sequence $(\sqcup_{i=0}^{k'} T^i)_{k' \geq 0}$ stabilizes for some $k' \geq 0$, i.e. $T^j \sqsubseteq \sqcup_{i=0}^{k'} T^i$ for all $j \geq k'$. Take the smallest such k'. For all $j \geq k'$ we have $T^{j+1} \sqsupseteq \text{post}^\# T^j$ and thus $\text{post}^\# T^j \sqsubseteq \sqcup_{i=0}^{k'} T^i$. By definition of the algorithm, the sequence $(T^j)_{j \geq k'}$ is monotonic increasing. The domain is finite, so there is a $k \geq k'$ with $T^{k+1} = T^k$.

Now we derive the number of iteration steps. The set $D^\#$ has chains of length at most $nGL + 1$, so in the increasing sequence $(\sqcup_{i=0}^{j} T^i)_{j \geq 0}$ there are at most nGL positions with a strict increase. Let us take two such "neighbour" positions $a < b$ with

$$\sqcup_{i=0}^{a-1} T^i \neq \sqcup_{i=0}^{a} T^i \text{ and } \sqcup_{i=0}^{b-1} T^i \neq \sqcup_{i=0}^{b} T^i$$

so that for all c with $a < c < b$ we have

$$\sqcup_{i=0}^{c-1} T^i = \sqcup_{i=0}^{c} T^i \ .$$

The sequence $(T^c)_{a \leq c < b}$ is increasing by the definition of the relaxed frontier sequence. The maximal chain length gives $b - a \leq nGL + 1$ and finally $k - k' \leq nGL$. Each of at most nGL positions with a strict increase in the sequence $(\sqcup_{i=0}^{j} T^i)_j$ can be followed by at most nGL positions with a strict increase of T^j before the next increase of $\sqcup_{i=0}^{j} T^i$ or termination. So the number of all increases is at most $nGL(nGL+1) = O((nGL)^2)$.

Now we show that the elements of the sequence up to position k represent a superset of states reachable from the initial ones.

Proposition 19. *The set* $X := \cup_{i=0}^{k} \gamma_{\text{cart}} T^i$ *is an inductive invariant, i.e.*

$$\text{init} \subseteq X \quad \textit{and} \quad \text{post} X \subseteq X \ .$$

Proof. Recall that $\gamma_{\text{cart}}\alpha_{\text{cart}}$ is extensive (i.e. for any $S \in D$ we get $S \subseteq \gamma_{\text{cart}}\alpha_{\text{cart}} S$). So the initial states are in X since $\text{init} \subseteq \gamma_{\text{cart}}\alpha_{\text{cart}}\text{init} = \gamma_{\text{cart}} T^0 \subseteq X$.
Now let $s \in \text{post}X$. Since post distributes over union, there is an i with $0 \leq i \leq k$ so that $s \in \text{post}\gamma_{\text{cart}} T^i$. Since $\gamma_{\text{cart}}\alpha_{\text{cart}}$ is extensive, we get $s \in \gamma_{\text{cart}}\alpha_{\text{cart}}\text{post}\gamma_{\text{cart}} T^i$ $= \gamma_{\text{cart}}\text{post}^\# T^i \subseteq \gamma_{\text{cart}} T^{i+1} \subseteq \cup_{i=1}^{k+1} \gamma_{\text{cart}} T^i$. Since $T^{k+1} \sqsubseteq T^k$, we get $\gamma_{\text{cart}} T^{k+1} \subseteq \gamma_{\text{cart}} T^k$ and thus $s \in \cup_{i=1}^{k} \gamma_{\text{cart}} T^i \subseteq X$.

An algorithm that implements relaxed frontier search in linear space in quadratic number of iterations is given below.

Algorithm 20 (Relaxed Frontier Search). *The input is* init *(the initial states) and f (an error state).*

T := $\bot$;   T' := α_{cart}init;
AllT := T';
while (T' $\not\sqsubseteq$ T $\wedge$ $\alpha_{\text{cart}}\{f\} \not\sqsubseteq$ T') do
    T := T';
    P := $\text{post}^{\#}$T;
    if P $\sqsubseteq$ AllT then T' := T $\sqcup$ P; else T' := P;
    AllT := AllT $\sqcup$ T';
od;
if $\alpha_{\text{cart}}\{f\} \not\sqsubseteq$T' then print ''program safe''
else print ''don't know'';

The runtime is polynomial, since $\text{post}^{\#}$ can be computed by the $\text{Post}^{\#}_{ij}$ rule and the number of iterations is quadratic in the number of threads. The used space is dominated by the size of AllT, which can be stored in $O(nGL)$ space, which is linear in the number of threads.

Now we look at the example 15 and show that the relaxed frontier search method is able to handle it. Below we derive the relaxed frontier sequence:

i	elements of the first component of T^i	elements of the second component of T^i
0	$(0, A)$	$(0, E)$
1	$(1, A)$	$(1, F)$
2	$(1, B)$, $(0, A)$	$(1, F)$, $(0, G)$
3	$(0, B)$	$(0, G)$
4	$(0, C)$	$(0, G)$

The elements after position $k = 4$ are all $((0, C), (0, G))$

The thread states $(\_, D)$ are not in T^i for any i. In this special example, the computation was equivalent to that of the pure frontier search, since on each step a new thread state was discovered. This will be the case for linear loop-free programs.

For the binary counter example 13, the computed invariant is equal to that of the pure Flanagan-Qadeer method, since the latter contains exactly the reachable states. However, the number of iterations is greater.

Quadratic Thread-Modular Method Now we show a technique based on local Cartesian abstraction that is able to prove mutual exclusion in some of those cases where the original algorithm was too coarse. We overcome the inherent coarseness of the abstraction by walking along chains in the bigger lattice $(D^{\#})^n$ with product ordering. The longest chains in the new domain have length $O(n^2GL)$. So the price we pay for proving mutual exclusion is squared complexity in comparison to the Flanagan-Qadeer algorithm.

Our algorithm maintains n^2 sets $\mathcal{R}_i^j \subseteq \text{Glob} \times \text{Loc}_\text{i}$ $(1 \leq i, j \leq n)$. The sets are generated by the following rules.

QInit:

$$\overline{\mathrm{init}_i \subseteq \mathcal{R}_i^j}$$

QStep:
$$\frac{(g, l_i) \in \mathcal{R}_i^j \qquad (g, l_i) \to_i (g', l_i') \qquad (g, l_k) \in \mathcal{R}_k^j}{(g', l_i') \in \mathcal{R}_i^i \qquad (g', l_k) \in \mathcal{R}_k^i} \; i \neq k$$

where init_i is the ith component of $\alpha_{\mathrm{cart}}\mathrm{init}$.
Let f be a user-given error state. If $\alpha_{\mathrm{cart}}\{f\} \sqsubseteq (\mathcal{R}_1^j, ..., \mathcal{R}_n^j)$ for some $1 \leq j \leq n$, then the algorithm says "don't know", otherwise the answer is "the program is safe".

Now we describe an intended meaning of the sets R_i^j $(1 \leq i, j \leq n)$. Let us view all the discovered thread states as partitioned into n tuples, namely $(\mathcal{R}_1^1, ..., \mathcal{R}_n^1), ..., (\mathcal{R}_1^n, ..., \mathcal{R}_n^n)$. Those thread states are added to the ith tuple which resulted due to a step of the ith thread. The QStep rule can be described informally as an iterative application of the following procedure:
- Take any tuple $(\mathcal{R}_1^j, ..., \mathcal{R}_n^j)$;
- Compute the successors that resulted due to a step of the ith thread;
- Add these successors to the ith tuple $(\mathcal{R}_1^i, ..., \mathcal{R}_n^i)$.

It is unclear why these rules should be sound and give a preciser answer than the original Flanagan-Qadeer rules. First we show

Theorem 21. *The set*

$$X := \bigcup_{j=1}^{n} \gamma_{\mathrm{cart}}(\mathcal{R}_1^j, ..., \mathcal{R}_n^j)$$

is an inductive invariant. Formally:

$$\mathrm{init} \subseteq X \quad \textit{and} \quad \mathrm{post}X \subseteq X.$$

Proof. We have $\mathrm{init} \subseteq \gamma_{\mathrm{cart}}\alpha_{\mathrm{cart}}\mathrm{init} \subseteq \gamma_{\mathrm{cart}}(R_1^j, ..., R_n^j) \subseteq X$ for any j, e.g. $j = 1$. Now take some program state $(g', l_1', ..., l_n') \in \mathrm{post}X$. We want to show that this state is in X. The successor operator post distributes over union, so there is an index $j \in \{1, ..., n\}$ so that $(g', l_1', ..., l_n') \in \mathrm{post}\gamma_{\mathrm{cart}}(R_1^j, ..., R_n^j)$. By definition of the successor map there is a thread $i \in \{1, ..., n\}$ which made a step from some state $(g, l_1, ..., l_n) \in \gamma_{\mathrm{cart}}(R_1^j, ..., R_n^j)$ so that $(g, l_i) \to_i (g', l_i')$ and for $k \neq i$ the other local parts remain the same, i.e. $l_k = l_k'$ for $k \neq i$. By definition of γ_{cart} for all $k \in \{1, ..., n\}$ holds $(g, l_k) \in \mathcal{R}_k^j$. Especially $(g, l_i) \in \mathcal{R}_i^j$, so by the QStep rule we have $(g', l_i') \in \mathcal{R}_i^i$ and $(g', l_k') = (g', l_k) \in \mathcal{R}_k^i$ for $k \neq i$. So for all $k \in \{1, ..., n\}$ we have $(g', l_k') \in \mathcal{R}_k^i$. By definition of the concretization map $(g', l_1', ..., l_n') \in \gamma_{\mathrm{cart}}(\mathcal{R}_1^i, ..., \mathcal{R}_n^i) \subseteq X$.

Before proving that this method is at least as precise as the method of Flanagan-Qadeer, recall that the Flanagan-Qadeer system of inference rules is equivalent to the following:

INIT:

$$\frac{}{\mathrm{init}_i \subseteq \mathcal{R}_i}$$

POST$^{\#}$:

$$\frac{(g,l_i) \in \mathcal{R}_i \qquad (g,l_i) \rightarrow_i (g',l_i') \qquad (g,l_k) \in \mathcal{R}_k}{(g',l_i') \in \mathcal{R}_i \qquad (g',l_k) \in \mathcal{R}_k}\ i \neq k.$$

Proposition 22. *The quadratic thread-modular method is at least as precise as the Flanagan-Qadeer method. Especially, for all threads $i \in \{1,...,n\}$ holds*

$$\bigcup_{j=1}^{n} \mathcal{R}_i^j \subseteq \mathcal{R}_i\ .$$

Proof. View the inference rules as rules that take the sets R_i^j ($1 \leq i,j \leq n$ for the quadratic thread-modular method) and R_i ($1 \leq i \leq n$ for the Flanagan-Qadeer method) in the premises and produce bigger sets $R_i'^j$ resp. R_i' in the conclusion. We use structural induction. We assume that $\cup_{j=1}^{n} R_i^j \subseteq R_i$ holds for the sets in the premises of the rules QINIT and QSTEP and show that this statement holds after applying each rule, i.e. $\cup_{j=1}^{n} R_i'^j \subseteq R_i' \quad (1 \leq i \leq n)$.
First consider the rule QINIT. From $R_i'^j = \mathrm{init}_i \cup R_i^j$ follows

$$\bigcup_{j=1}^{n} R_i'^j = \mathrm{init}_i \cup \bigcup_{j=1}^{n} R_i^j \subseteq \mathrm{init}_i \cup R_i = R_i'.$$

Now consider the rule QSTEP. Let $(g',l_i') \in R_i'^i$ for some $i \in \{1,...,n\}$. If this element is in R_i^i, it is also in R_i by induction assumption. Otherwise it is added by the rule. Then there is a thread state $(g,l_i) \in R_i^j$ for some j with $(g,l_i) \rightarrow_i (g',l_i')$. By induction assumption $(g,l_i) \in R_i$ and by the POST$^{\#}$ rule $(g',l_i') \in R_i'$. So $R_i'^i \subseteq R_i'$.
Now let $(g',l_k) \in R_k'^i$ for $k \neq i$ from $\{1,...,n\}$. If (g',l_k) is in R_k^i, it is also in R_k by induction assumption. Otherwise it is added by the rule. Then there is an index $j \in \{1,..,n\}$ and a thread state $(g,l_i) \in R_i^j$ with $(g,l_i) \rightarrow_i (g',\_)$ so that $(g,l_k) \in R_k^j$. By induction assumption $(g,l_i) \in R_i$, $(g,l_k) \in R_k$ and by the POST$^{\#}$ rule $(g',l_k) \in R_k'$. So $R_k'^i \subseteq R_k'$.

Now let us turn to the mutual exclusion example 16. Now we show a broader class of programs that can now be handled by the quadratic thread-modular method.

Example 23. For any n, consider a program that consists of a single global variable m with initial value 0 and n copies of the following thread:

```
A: <wait until m=0;
       m:=1;>
B: m:=0;
C:
```

The quadratic thread-modular algorithm generates the following sets:
$\mathcal{R}_j^j = \{(0, A), (1, B), (0, C)\}$ for $j \in \{1, ..., n\}$,
$\mathcal{R}_i^j = \{(0, A), (1, A), (0, C), (1, C)\}$ for $i \neq j$ in $\{1, ..., n\}$.

For each $j \in \{1, ..., n\}$, only one set from $\mathcal{R}_1^j, ..., \mathcal{R}_n^j$ contains a thread state $(\_, B)$, namely, $(1, B) \in \mathcal{R}_j^j$. So the invariant $\cup_{j=1}^n \gamma_{\text{cart}}(\mathcal{R}_1^j, ..., \mathcal{R}_n^j)$ contains only states with at most one thread in critical section at label B.

It is not immediately clear why mutual exclusion can be proven in the above example. If you look at the QStep rule, you see that whatever value the index j has, the rule puts the thread states that resulted due to a step of the thread i into the sets $\mathcal{R}_i^i$ and $\mathcal{R}_k^i$ $(k \neq i)$. So whenever a program enters the critical section of the thread i, the abstraction of the corresponding program state is in $\mathcal{R}_1^i, ..., \mathcal{R}_n^i$. Whenever another thread i' enters its critical section, the abstraction of the corresponding program state goes into $\mathcal{R}_1^{i'}, ..., \mathcal{R}_n^{i'}$. The concretization of each tuple $(\mathcal{R}_1^i, ..., \mathcal{R}_n^i)$ is done separately for each thread i (see Theorem 21), so states of one thread in its critical section are not mixed with states of another thread in its critical section.

An implementation of the quadratic thread-modular method needs $O(n^2GL)$ space and polynomial time. The exact runtime depends on how the sets $\mathcal{R}_i^j$ and the transition relation are represented. One can consider implementing the rule QStep in the assume-guarantee style of the original algorithm by introducing auxiliary sets $\mathcal{G}_i^j \subseteq \text{Glob}^2$ that contain changes of the global variables induced by thread states in $\mathcal{R}_i^j \quad (1 \leq i, j \leq n)$.

8.3 Devising new algorithms

The presented two techniques of relaxed frontier search and quadratic thread-modular method can be combined gaining even more precision while producing a polynomial-time algorithm. This is left as an exercise for the reader.

Now consider the lattice $(D^{\#})^{p(n)}$ with product ordering where $p(n)$ is some polynomial. Its longest chains have $p(n)nGL + 1$ elements. Any algorithm that iteratively produces elements of an ascending chain would inevitably need polynomial number of iterations. It would be interesting to investigate the ability of proving results by walking along chains in these domains.

9 Summary

We have examined an approach for verifying concurrent programs.
On the one hand, we have examined the Flanagan-Qadeer algorithm for checking safety of multithreaded programs. We have characterized it in a well-known framework of abstract interpretation. Using this characterization, we have shown the boundary of this algorithm.
On the other hand, we have started developing the theory of Cartesian abstraction for multithreaded programs. We have shown two equivalent approaches for abstract fixpoint checking on the abstract and the concrete domain. We have

seen that local Cartesian abstraction is polynomial in the number of threads. Using ideas from both points of view, we have devised two new polynomial-time algorithms based on Cartesian abstraction that substantially increase precision of the thread-modular method.
These contributions seem to be first steps in a systematic study of similar abstractions of the state explosion problem.

10 Acknowledgements

We would like to thank anonymous reviewers for their constructive comments and suggestions.

References

1. Bruno Blanchet. Introduction to abstract interpretation, 2002. Lecture script, `http://bblanche.gitlabpages.inria.fr/absint.pdf`.
2. Patrick Cousot. Partial completeness of abstract fixpoint checking. In Berthe Y. Choueiry and Toby Walsh, editors, *SARA*, volume 1864 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2000.
3. Patrick Cousot and Radhia Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Math.*, 82(1):43–57, 1979.
4. Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282, 1979.
5. Cormac Flanagan and Shaz Qadeer. Thread-modular model checking. In Thomas Ball and Sriram K. Rajamani, editors, *SPIN*, volume 2648 of *Lecture Notes in Computer Science*, pages 213–224. Springer, 2003.
6. Thomas Anton Henzinger, Ranjit Jhala, Rupak Majumdar, and Shaz Qadeer. Thread-modular abstraction refinement. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 262–274. Springer, 2003.
7. Gerard J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
8. Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 261–262. IEEE, 1977.
9. Patrick Maier. A framework for circular assume-guarantee reasoning, 2002. Symposium on the Effectiveness of Logic in Computer Science in Honour of Moshe Vardi, research report MPI-I-2002-2-007.
10. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
11. Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety.* Springer-Verlag, May 1995.