On the Complexity of Bounded Context Switching[*]

Peter Chini[1], **Jonathan Kolberg**[2], **Andreas Krebs**[†3], **Roland Meyer**[‡4], and **Prakash Saivasan**[5]

1 **TU Braunschweig, Germany**
`p.chini@tu-bs.de`
2 **TU Braunschweig, Germany**
`j.kolberg@tu-bs.de`
3 **Universität Tübingen, Germany**
`krebs@informatik.uni-tuebingen.de`
4 **TU Braunschweig, Germany**
`roland.meyer@tu-bs.de`
5 **TU Braunschweig, Germany**
`p.saivasan@tu-bs.de`

Abstract

Bounded context switching (BCS) is an under-approximate method for finding violations to safety properties in shared-memory concurrent programs. Technically, BCS is a reachability problem that is known to be NP-complete. Our contribution is a parameterized analysis of BCS.

The first result is an algorithm that solves BCS when parameterized by the number of context switches (cs) and the size of the memory (m) in $\mathcal{O}^*(m^{cs} \cdot 2^{cs})$. This is achieved by creating instances of the easier problem Shuff which we solve via fast subset convolution. We also present a lower bound for BCS of the form $m^{o(cs/\log(cs))}$, based on the exponential time hypothesis. Interestingly, the gap is closely related to a conjecture that has been open since FOCS'07. Further, we prove that BCS admits no polynomial kernel.

Next, we introduce a measure, called scheduling dimension, that captures the complexity of schedules. We study BCS parameterized by the scheduling dimension ($sdim$) and show that it can be solved in $\mathcal{O}^*((2m)^{4sdim}4^t)$, where t is the number of threads. We consider variants of the problem for which we obtain (matching) upper and lower bounds.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Shared memory concurrency, safety verification, fixed-parameter tractability, exponential time hypothesis, bounded context switching

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.27

1 Introduction

Concurrent programs where several threads interact through a shared memory can be found essentially everywhere where performance matters, in particular in critical infrastructure like operating systems and libraries. The asynchronous nature of the communication makes these

[*] A full version of the paper is available at `https://arxiv.org/abs/1609.09728`.
[†] Supported by the DFG Emmy Noether program (KR4042/2).
[‡] A part of the work was carried out when the author was at Aalto University.

programs prone to programming errors. As a result, substantial effort has been devoted to developing automatic verification tools. The current trend for shared memory is bug-hunting: Algorithms that look for misbehavior in an under-approximation of the computations.

The most prominent method in the under-approximate verification of shared-memory concurrent programs is bounded context switching (BCS) [48]. A context switch occurs when a thread leaves the processor for another thread to be scheduled. The idea of BCS is to limit the number of times the threads may switch the processor. Effectively this limits the communication that can occur between the threads. (Note that there is no bound on the running time of each thread.) Bounded context switching has received considerable attention [37, 4, 3, 1, 38, 39, 2, 47] for at least two reasons. First, the under-approximation has been demonstrated to be useful in numerous experiments, in the sense that synchronization bugs show up in few context switches [46]. Second, compared to other verification methods, BCS is algorithmically appealing, with the complexity dropping from PSPACE to NP in the case of Boolean programs.

The hardness of verification problems, also the NP-hardness of BCS, is in sharp contrast to the success that verification tools see on industrial instances. This discrepancy between the worst-case behavior and efficiency in practice has also been observed in other areas within algorithmics. The response was a line of research that refines the classical worst-case complexity. Rather than only considering problems where the instance-size determines the running time, so-called parameterized problems identify further parameters that give information about the structure of the input or the shape of solutions. The complexity class of interest consists of the so-called fixed-parameter tractable problems. A problem is fixed-parameter tractable if the parameter that has been identified is indeed responsible for the non-polynomial running time or, phrased differently, the running time is $f(k)p(n)$ where k is the parameter, n is the size of the input, f is a computable function, and p is a polynomial.

Within fixed-parameter tractability, the recent trend is a fine-grained analysis to understand the precise functions f that are needed to solve a problem. From an algorithmic point of view, an exponential dependence on k, at best linear so that $f(k) = 2^k$, is particularly attractive. There are, however, problems where algorithms running in $2^{o(k\log(k))}$ are unlikely to exist. As common in algorithmics, unconditional lower bounds are hard to achieve, and none are known that separate 2^k and $2^{k\log(k)}$. Instead, one works with the so-called exponential time hypothesis (ETH): After decades of attempts, n-variable 3-SAT is not believed to admit an algorithm of running time $2^{o(n)}$. To derive a lower bound for a problem, one now shows a reduction from n-variable 3-SAT to the problem such that a running time in $2^{o(k\log(k))}$ means ETH breaks.

The contribution of our work is a fine-grained complexity analysis of the bounded context switching under-approximation. We propose algorithms as well as matching lower bounds in the spectrum 2^k to k^k. This work is not merely motivated by explaining why verification works in practice. Verification tasks have also been shown to be hard to parallelize. Due to the memory demand, the current trend in parallel verification is lock-free data structures [6]. So far, GPUs have not received much attention. With an algorithm of running time $2^k p(n)$, and for moderate k, say 12, one could run in parallel 4096 threads each solving a problem of polynomial effort.

When parameterized only by the context switches, BCS is quickly seen to be W[1]-hard and hence does not admit an FPT-algorithm. Since it is often the case that shared memory communication is via signaling (flags), the memory requirements are not high. We additionally parameterize by the memory. Our study can be divided into two parts.

We first give a parameterization of BCS (in the context switches and the size of the memory) that is *global* in the sense that all threads share a budget of cs many context switches. For the upper bound, we show that the problem can be solved in $\mathcal{O}^*(m^{cs}2^{cs})$. We first enumerate the sequences of memory states at which the threads could switch context, and there are m^{cs} such sequences where m is the size of the memory. For a given such sequence, we check a problem called Shuff: Do the threads have computations that justify the sequence (and lead to their accepting state)? Here, we use fast subset convolution to solve Shuff in $\mathcal{O}^*(2^{cs})$. Note that Shuff is a problem that may be interesting in its own right. It is an under-approximation that still leaves much freedom for the local computations of the threads. Indeed, related ideas have been used in testing [33, 12, 24, 31].

For the lower bound, the finding is that the global parameterization of BCS is closely related to *Subgraph Isomorphism* (SGI). Whereas the reduction is not surprising, the relationship is, with SGI being one of the problems whose fine-grained complexity is not fully understood. Subgraph isomorphism can be solved in $\mathcal{O}^*(n^k)$ where k is the number of edges in the graph that is to be embedded. The only lower bound, however, is $n^{o(k/\log k)}$, and has, to the best of our knowledge, not been improved since FOCS'07 [44, 45]. However, the belief is that the $\log k$-gap in the exponent can be closed. We show how to reduce SGI to the global version of BCS, and obtain an $m^{o(cs/\log cs)}$ lower bound. Phrased differently, BCS is harder than SGI but admits the same upper bound. So once Marx' conjecture is proven, we obtain a matching bound. If we proved a lower upper bound, we had disproven Marx' conjecture.

Our second contribution is a study of BCS where the parameterization is *local* in the sense that every thread is given a budget of context switches. Here, our focus is on the scheduling. We associate with computations so-called scheduling graphs that show how the threads take turns. We define the scheduling dimension, a measure on scheduling graphs (shown to be closely related to carving width) that captures the complexity of a schedule. Our main finding is a fixed-point algorithm that solves the local variant of BCS exponential only in the scheduling dimension and the number of threads. We study variants where only the budget of context switches is given, the graph is given, and where we assume round robin as a schedule. Verification under round robin has received quite some attention [10, 46, 40]. In that setting, we show that we get rid of the exponential dependence on the number of threads and obtain an $\mathcal{O}^*(m^{4cs})$ upper bound. We complement this by a matching lower bound.

The following table summarizes our results and highlights the main findings in gray.

Problem	Upper Bound	Lower Bound
Shuff	$\mathcal{O}^*(2^k)$	$(2-\varepsilon)^k$
BCS	$\mathcal{O}^*(m^{cs}2^{cs})$	$m^{o(cs/\log cs)}$, no poly. kernel
BCS-L-RR	$\mathcal{O}^*(m^{4cs})$	$2^{o(cs\log(m))}$
BCS-L-FIX	$\mathcal{O}^*((2m)^{4sdim})$	$2^{o(sdim\log(m))}$
BCS-L	$\mathcal{O}^*((2m)^{4sdim}4^t)$	$2^{o(sdim\log(m))}$

The organization is by expressiveness, measured in terms of the amount of computations that an analysis explores. Considering shuffle membership Shuff as an under-approximate analysis in its own right, Shuff is less expressive than the globally parameterized BCS. BCS is less expressive than round robin BCS-L-RR, which is an instance of fixing the scheduling graph BCS-L-FIX. The most liberal parameterization is via the scheduling dimension BCS-L. In the paper, we present algorithms for the case where the threads are finite state. Our results also hold for more general classes of programs, notably recursive ones. The only condition that we require is that the chosen automaton model for the threads has a polynomial time decision procedure for checking non-emptiness when intersected with a regular language.

There have been previous efforts in studying fixed-parameter tractable algorithms for automata and verification-related problems. In [21], the authors introduced the notion of conflict serializability under TSO and gave an FPT-algorithm for checking serializability. In [24], the authors studied the complexity of predicting atomicity violation on concurrent systems and showed that no FPT solution is possible for the same. In [18], various model checking problems for synchronized executions on parallel components were considered and proven to be intractable. Parameterized complexity analyses for different problems on finite automata were given in [25, 26, 50].

Verification of concurrent systems has received considerable attention. The parameterized verification was studied in [20, 22, 29, 34, 38]. Concurrent shared-memory systems with a fixed number of threads were also studied in [2, 3, 5].

2 Preliminaries

We define the bounded context switching problem of interest [48] and recall the basics on fixed-parameter tractability following [19, 27].

Bounded Context Switching. We study the safety verification problem for shared-memory concurrent programs. To obtain precise complexity results, it is common to assume both the number of threads and the data domain to be finite. Safety properties partition the states of a program into *unsafe* and *safe* states. Hence, checking safety amounts to checking whether no unsafe state is reachable. In the following, we develop a language-theoretic formulation of the reachability problem that will form the basis of our study.

We model the shared memory as a (non-deterministic) finite automaton of the form $M = (Q, \Sigma, \delta_M, q_0, q_f)$. The states Q correspond to the data domain, the set of values that the memory can be in. The initial state $q_0 \in Q$ is the value that the computation starts from. The final state $q_f \in Q$ reflects the reachability problem. The alphabet Σ models the set of operations. Operations have the effect of changing the memory valuation, formalized by the transition relation $\delta_M \subseteq Q \times \Sigma \times Q$. We generalize the transition relation to words $u \in \Sigma^*$. The set of sequences of operations that lead from a state q to another state q' is the language $L(M(q, q')) := \{u \in \Sigma^* \mid q' \in \delta_M(q, u)\}$. The language of M is $L(M) := L(M(q_0, q_f))$. The size of M, denoted $|M|$, is the number of states.

We also model the threads operating on the shared memory M as finite automata $A_{id} = (P, \Sigma \times \{id\}, \delta_A, p_0, p_f)$. Note that they use the alphabet Σ of the shared memory, indexed by the name of the thread. The index will play a role when we define the notion of context switches below. The automaton A_{id} is nothing but the control flow graph of the thread id. Its language is the set of sequences of operations that the thread could potentially execute to reach the final state. As the thread language does not take into account the effect of the operations on the shared memory, not all these sequences will be feasible. Indeed, the thread may issue a command $\mathit{write}(x, 1)$ followed by $\mathit{read}(x, 0)$, which the automaton for the shared memory will reject. The computations of A that are actually feasible on the shared memory are given by the intersection $L(M) \cap L(A_{id})$. Here, we silently assume the intersection to project away the second component of the thread alphabet.

A concurrent program consists of multiple threads A_1 to A_t that mutually influence each other by accessing the same memory M. We mimic this influence by interleaving the thread languages, formalized with the shuffle operator $\sqcup\!\sqcup$. Consider languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ over disjoint alphabets $\Sigma_1 \cap \Sigma_2 = \emptyset$. The shuffle of the languages contains all words over the union of the alphabets where the corresponding projections $(- \downarrow -)$ belong to the operand languages, $L_1 \sqcup\!\sqcup L_2 := \{u \in (\Sigma_1 \cup \Sigma_2)^* \mid u \downarrow \Sigma_i \in L_i \cup \{\varepsilon\}, i = 1, 2\}$.

With these definitions in place, a *shared-memory concurrent program (SMCP)* is a tuple $S = (\Sigma, M, (A_i)_{i\in[1..t]})$. Its language is $L(S) := L(M) \cap (\ ⧢_{i\in[1..t]} L(A_i)\ )$. The safety verification problem induced by the program is to decide whether $L(S)$ is non-empty. Note that we use $[1..t]$ to identify the set $\{1,\ldots,t\}$.

We formalize the notion of context switching. Every word in the shuffle of the thread languages, $u \in ⧢_{i\in[1..t]} L(A_i)$, has a unique decomposition into maximal infixes that are generated by the same thread. Formally, $u = u_1 \ldots u_{cs+1}$ so that there is a function $\varphi : [1..cs+1] \to [1..t]$ satisfying $u_i \in (\Sigma \times \{\varphi(i)\})^+$ and $\varphi(i) \neq \varphi(i+1)$ for all $i \in [1..cs]$. We refer to the u_i as contexts and to the thread changes between u_i to u_{i+1} as *context switches*. So u has $cs+1$ contexts and cs context switches. Let $\text{Context}(\Sigma, t, cs)$ denote the set of words (over Σ with t threads) that have at most cs-many context switches. The *bounded context switching* under-approximation limits the safety verification task to this language.

Bounded Context Switching (BCS)
Input: An SMCP $S = (\Sigma, M, (A_i)_{i\in[1..t]})$ and a bound $cs \in \mathbb{N}$.
Question: Is $L(S) \cap \text{Context}(\Sigma, t, cs) \neq \emptyset$?

Fixed Parameter Tractability. BCS is NP-complete, even for unary alphabets [23]. Our goal is to understand which instances can be solved efficiently and, in turn, what makes an instance hard. Parameterized complexity addresses these questions.

A *parameterized problem* L is a subset of $\Sigma^* \times \mathbb{N}$. The problem is *fixed-parameter tractable (FPT)* if there is a deterministic algorithm that, given $(x,k) \in \Sigma^* \times \mathbb{N}$, decides $(x,k) \in L$ in time $f(k) \cdot |x|^{O(1)}$. Here, f is a computable function that only depends on the parameter k. It is common to denote the running time by $\mathcal{O}^*(f(k))$ and suppress the polynomial part.

While many parameterizations of NP-hard problems were proven to be fixed-parameter tractable, there are problems that are unlikely to be FPT. A famous example that we shall use is k-Clique, the problem of finding a clique of size k in a given graph. k-Clique is complete for the complexity class W[1], and W[1]-hard problems are believed to lie outside FPT.

A theory of relative hardness needs an appropriate notion of reduction. Given parameterized problems $L, L' \subseteq \Sigma^* \times \mathbb{N}$, we say that L is *reducible* to L' via a *parameterized reduction*, denoted by $L \leq^{fpt} L'$, if there is an algorithm that transforms an input (x,k) to an input (x',k') in time $g(k) \cdot n^{O(1)}$ so that $(x,k) \in L$ if and only if $(x',k') \in L'$. Here, g is a computable function and k' is computed by a function only dependent on k.

For BCS, a first result is that a parameterization by the number of context switches and additionally by the number of threads, denoted by BCS(cs,t), is not sufficient for FPT: The problem is W[1]-hard. It remains in W[1] if we only parameterize by the context switches.

▶ **Proposition 1.** BCS(cs) *and* BCS(cs,t) *are both* W[1]*-complete.*

The running time of an FPT-algorithm is dominated by f. The goal of *fine-grained complexity theory* is to give upper and lower bounds on this non-polynomial function. For lower bounds, the problem that turned out to be hard is n-variable 3-SAT. The *Exponential Time Hypothesis* (ETH) is that n-variable 3-SAT does not admit a $2^{o(n)}$-time algorithm [36]. We will prove a number of lower bounds that hold, provided ETH is true.

In the remainder of the paper, we consider parameterizations of BCS that are FPT. Our contribution is a fine-grained complexity analysis.

3 Global Parameterization

Besides the number of context switches cs, we now consider the size m of the memory as a parameter of BCS. This parameterization is practically relevant and, as we will show, algorithmically appealing. Concerning the relevance, note that communication over the shared memory is often implemented in terms of flags. Hence, when limiting the size of the memory we still explore a large part of the computations.

Upper Bounds. The idea of our algorithm is to decompose BCS into exponentially many instances of the easier problem shuffle membership (Shuff) defined below. Then we solve Shuff with fast subset convolution. To state the result, let the given instance of BCS be $S = (\Sigma, M, (A_i)_{i\in[1..t]})$ with bound cs. To each automaton A_i, our algorithm will associate another automaton B_i of size polynomial in A_i. Let $b = \max_{i\in[1..t]} |B_i|$. Moreover, let $\mathsf{Shuff}(b, k, t) = \mathcal{O}(2^k \cdot t \cdot k \cdot (b^2 + k \cdot bc(k)))$ be the complexity of solving the shuffle problem. The factor $bc(k)$ appears as we need to multiply k-bit integers (see below). The currently best known running time is $bc(k) = k \log k \cdot 2^{\mathcal{O}(\log^* k)}$ [32, 35].

▶ **Theorem 2.** BCS *can be solved in* $\mathcal{O}(m^{cs+1} \cdot \mathsf{Shuff}(b, cs + 1, t) + t \cdot m^3 \cdot b^3)$.

We decompose BCS along *interface sequences*. Such an interface sequence is a word $\sigma = (q_1, q_1') \dots (q_k, q_k')$ over pairs of states of the memory automaton M. The length is k. An interface sequence is *valid* if q_1 is the initial state of the memory automaton, q_k' the final state, and $q_i' = q_{i+1}$ for $i \in [1..k-1]$. Consider a word $u \in L(S)$ with contexts $u = u_1 \dots u_m$. An interface sequence $\sigma = (q_0, q_1)(q_1, q_2) \dots (q_{m-1}, q_m)$ is *induced* by u, if there is an accepting run of M on u such that for all $i \in [1..m]$, q_i is the state reached by M upon reading $u_1 \dots u_i$. Note that we only consider the states that occur upon context switches. Moreover, induced sequences are valid by definition. Finally, note that a word with cs-many context switches induces an interface sequence of length precisely $cs + 1$. We define $\mathit{IIF}(S) \subseteq (Q \times Q)^*$ to be the *language of all induced interface sequences.*

Induced interface sequences witness non-emptiness of $L(S)$: $L(S) \neq \emptyset$ iff $\mathit{IIF}(S) \neq \emptyset$. Since the number of context switches is bounded by cs, we can thus iterate over all sequences in $(Q \times Q)^{\leq cs+1}$ and test each of them for being an induced interface sequence, i.e. an element of $\mathit{IIF}(S)$. Since induced sequences are valid, there are at most m^{cs+1} sequences to test.

Before turning to this test, we do a preprocessing step that removes the dependence on the memory automaton M. To this end, we define the *interface language* $\mathit{IF}(A_{id})$ of a thread. It makes visible the state changes on the shared memory that the contexts of this thread may induce. Formally, the interface language consists of all interface sequences $(q_1, q_1') \dots (q_k, q_k')$ so that $L(A_{id}) \cap (\ L(M(q_1, q_1')) \dots L(M(q_k, q_k'))\ ) \neq \emptyset$. These sequences do not have to be valid as the thread may be interrupted by others. Below, we rely on the fact that $\mathit{IF}(A_{id})$ is again a regular language, a representation of which is easy to compute.

▶ **Lemma 3.**
(i) *We have* $\mathit{IIF}(S) = ⧢_{i\in[1..t]} \mathit{IF}(A_i) \cap \{\sigma \in (Q \times Q)^* \mid \sigma \text{ valid}\}$.
(ii) *One can compute in time* $\mathcal{O}(|A_{id}|^3 \cdot |M|^3)$ *an automaton* B_{id} *with* $L(B_{id}) = \mathit{IF}(A_{id})$.

It remains to check whether a valid sequence $\sigma \in (Q \times Q)^{cs+1}$ is included in the shuffle $⧢_{i\in[1..t]} L(B_i)$. This means we address the following problem:

Shuffle Membership (Shuff)
Input: NFAs $(B_i)_{i\in[1..t]}$ over the alphabet Γ, an integer k, and a word $w \in \Gamma^k$.
Question: Is w in $Ш_{i\in[1..t]} L(B_i)$?

We obtain the following upper bound, with b and $bc(k)$ as defined above.

▶ **Theorem 4.** Shuff *can be solved in time* $\mathcal{O}(2^k \cdot t \cdot k \cdot (b^2 + k \cdot bc(k)))$.

Our algorithm is based on *fast subset convolution* [7], an algebraic technique for summing up partitions of a given set. Typically, fast subset convolution is applied to graph problems: Björklund et al. [7] used it to present the first $\mathcal{O}^*(2^k)$-time algorithm for the Steiner Tree problem with k terminals and bounded edge weights. Cygan et al. incorporated a generalized version as a subprocedure in applications of their *Cut & Count* technique [17]. Variants of Dominating Set parameterized by treewidth were solved by van Rooij et al. in [49] using fast subset convolution. We are not aware of an automata-theoretic application.

Let $f, g : \mathcal{P}(B) \to \mathbb{Z}$ be two functions from the powerset of a k-element set B to the ring of integers. The *convolution* of f and g is the function $f * g : \mathcal{P}(B) \to \mathbb{Z}$ that maps a subset $S \subseteq B$ to the sum $\sum_{U \subseteq S} f(U)g(S \setminus U)$. Note that the convolution is associative. There is a close connection to partitions. For $t \in \mathbb{N}$, a *t-partition* of a set S is a tuple $(U_1, \ldots, U_t)$ of subsets of S such that $U_1 \cup \cdots \cup U_t = S$ and $U_i \cap U_j = \emptyset$ for all $i \neq j$. Now it is easy to see that the convolution of t functions $f_i : \mathcal{P}(B) \to \mathbb{Z}, i \in [1..t]$, sums up all t-partitions of S:

$$(f_1 * \cdots * f_t)(S) = \sum_{\substack{(U_1,\ldots,U_t) \\ \text{is a } t\text{-parition of } S}} f_1(U_1) \cdots f_t(U_t) \; .$$

To apply the convolution, we give a characterization of Shuff in terms of partitions. Let $((B_i)_{i\in[1..t]}, k, w)$ be an instance of Shuff. The following observation is crucial. The word w lies in the shuffle of the $L(B_i)$ if and only if there are non-overlapping, possibly empty (scattered) subwords $w_1, \ldots, w_t$ of w that decompose w and that satisfy $w_i \in L(B_i) \cup \{\varepsilon\}$ for all $i \in [1..t]$. By scattered, we mean that the subwords do not have to form an infix of w. Such a decomposition induces a t-partition $(U_1, \ldots, U_t)$ of the set of positions $\text{Pos} = \{1, \ldots, k\}$ of w, where each U_i holds exactly the positions of w_i. In turn, given a t-partition $(U_1, \ldots, U_t)$ of Pos, we can derive a decomposition of w by setting $w_i = w[U_i]$ for all $i \in [1..t]$. Here, $w[U_i]$ is the projection of w to the positions in U_i. Hence, w lies in the shuffle if and only if there is a t-partition $(U_1, \ldots, U_t)$ of Pos such that $w[U_i] \in L(B_i) \cup \{\varepsilon\}$ for all $i \in [1..t]$.

To express the language membership in $L(B_i)$ in terms of functions, we employ the characteristic functions $f_i : \mathcal{P}(\text{Pos}) \to \mathbb{Z}$ that map a set S to 1 if $w[S] \in L(B_i) \cup \{\varepsilon\}$, and to 0 otherwise. By the above formula, it follows that $(f_1 * \cdots * f_t)(\text{Pos}) > 0$ if and only if there is a t-partition $(U_1, \ldots, U_t)$ of Pos such that $f_i(U_i) = 1$ for $i \in [1..t]$. Altogether, we have proven the following lemma:

▶ **Lemma 5.** *The word* $w \in \Gamma^k$ *is in* $Ш_{i\in[1..t]} L(B_i)$ *if and only if* $(f_1 * \cdots * f_t)(\text{Pos}) > 0$.

Our algorithm for Shuff computes the characteristic functions f_i and $t-1$ convolutions to obtain $f_1 * \cdots * f_t$. Then it evaluates the convolution at the set Pos. Computing and storing a value $f_i(S)$ for a subset $S \subseteq \text{Pos}$ takes time $\mathcal{O}(k \cdot b^2)$ since we have to test membership of a word of length at most k in B_i. Hence, computing all f_i takes time $\mathcal{O}(2^k \cdot t \cdot k \cdot b^2)$. Due to Björklund et al. [7], we can compute the convolution of two functions $f, g : \mathcal{P}(\text{Pos}) \to \mathbb{Z}$ in $\mathcal{O}(2^k \cdot k^2)$ multiplications in $\mathbb{Z}$. Furthermore, if the ranges of f and g are bounded by C, we have to perform these operations on $\mathcal{O}(k \log C)$-bit integers [7]. Since the characteristic functions f_i have ranges bounded by a constant, we only need to compute with $\mathcal{O}(k)$-bit integers. Hence, the $t-1$ convolutions can be carried out in time $\mathcal{O}(2^k \cdot k^2 \cdot (t-1) \cdot bc(k))$. Altogether, this proves Theorem 4.

Lower Bound for Bounded Context Switching. We prove a lower bound for the NP-hard BCS by reducing the subgraph isomorphism problem (SGI) to it. The result is such that it also applies to BCS(cs) and BCS(cs, m). We explain why the result is non-trivial.

In fine-grained complexity, lower bounds for W[1]-hard problems are often obtained by reductions from k-Clique. Chen et al. [13] have shown that k-Clique cannot be solved in time $f(k)n^{o(k)}$ for any computable function f, unless ETH fails. To transport the lower bound to a problem of interest, one has to construct a parameterized reduction that blows up the parameter only linearly. In the case of BCS, this fails. We face a well-known problem which was observed for reductions using edge-selection gadgets [45, 16]: A reduction from k-Clique would need to select a clique candidate of size k and check whether every two vertices of the candidate share an edge. This needs $\mathcal{O}(k^2)$ communications between the chosen vertices, which translates to $\mathcal{O}(k^2)$ context switches. Hence, we only obtain $n^{o(\sqrt{k})}$ as a lower bound.

To overcome this, we follow Marx [45] and give a reduction from SGI. This problem takes as input two graphs G and H and asks whether G is isomorphic to a subgraph of H. This means that there is an injective map $\varphi : V(G) \to V(H)$ such that for each edge (u, v) in G, the pair $(\varphi(u), \varphi(v))$ is an edge in H. We use $V(G)$ to denote the vertices and $E(G)$ to denote the edges of a graph G. Marx has shown that SGI cannot be solved in time $f(k)n^{o(k/\log k)}$, where k is the number of edges of G, unless ETH fails. In our reduction, the number of edges is mapped linearly to the number of context switches.

▶ **Theorem 6.** *Assuming* ETH*, there is no f s.t.* BCS *can be solved in $f(cs)n^{o(cs/\log(cs))}$.*

Roughly, the idea is this: The alphabet $V(G) \times V(H)$ describes how the vertices of G are mapped to vertices of H. Now we can use the memory M to output all possible injective maps from $V(G)$ to $V(H)$. There is one thread A_i for each edge of G. Its task is to verify that the edges of G get mapped to edges of H.

Note that Theorem 6 implies a lower bound for the FPT-problem BCS(cs, m). It cannot be solved in time $m^{o(cs/\log(cs))}$, unless ETH fails.

Lower Bound for Shuffle Membership. We prove it unlikely that Shuff can be solved in $\mathcal{O}^*((2-\delta)^k)$ time, for a $\delta > 0$. Hence, the $\mathcal{O}^*(2^k)$-time algorithm above may be optimal. We base our lower bound on a reduction from Set Cover. An instance consists of a family of sets $(S_i)_{i\in[1..m]}$ over a universe $U = \bigcup_{i\in[1..m]} S_i$, and an integer $t \in \mathbb{N}$. The problem asks for t sets $S_{i_1}, \ldots, S_{i_t}$ from the family such that $U = \bigcup_{j\in[1..t]} S_{i_j}$.

We are interested in a parameterization of the problem by the size n of the universe. It was shown that this parameterization admits an $\mathcal{O}^*(2^n)$-time algorithm [28]. But so far, no $\mathcal{O}^*((2-\varepsilon)^n)$-time algorithm was found, for an $\varepsilon > 0$. Actually, the authors of [15] conjecture that the existence of such an algorithm would contradict the *Strong Exponential Time Hypothesis* (SETH) [36, 11]. This is the assumption that n-variable SAT cannot be solved in $\mathcal{O}^*((2-\varepsilon)^n)$ time, for an $\varepsilon > 0$ (SETH implies ETH). By now, there is a list of lower bounds based on Set Cover [8, 15]. We add Shuff to this list.

▶ **Proposition 7.** *If* Shuff *can be solved in time $\mathcal{O}^*((2-\delta)^k)$ for a $\delta > 0$, then* Set Cover *can be solved in time $\mathcal{O}^*((2-\varepsilon)^n)$ for an $\varepsilon > 0$.*

Lower Bound on the Size of the Kernel. Kernelization is a preprocessing technique for parameterized problems that transforms a given instance to an equivalent instance of size bounded by a function in the parameter. It is well-known that any FPT-problem admits a kernelization and any kernelization yields an FPT-algorithm [16]. The search for small problem kernels is ongoing research. A survey can be found in [42].

There is also the opposite approach, disproving the existence of a kernel of polynomial size [9, 30]. Such a result indicates hardness of the problem at hand, and hence serves as a lower bound. Technically, the existence of a polynomial kernel is linked to the inclusion NP $\subseteq$ coNP/poly. The latter is unlikely as it would cause a collapse of the polynomial hierarchy to the third level [51]. Based on this approach, we show that $\mathsf{BCS}(cs, m)$ does not admit a kernel of polynomial size. We introduce the needed notions, following [16].

A *kernelization* for a parameterized problem Q is an algorithm that, given an instance (I, k), returns an equivalent instance (I', k') in polynomial time such that $|I'| + k' \leq g(k)$ for some computable function g. If g is a polynomial, Q is said to admit a *polynomial kernel.*

We also need *polynomial equivalence relations.* These are equivalence relations on Σ^*, with Σ some alphabet, such that: (1) There is an algorithm that, given $x, y \in \Sigma^*$, decides whether $(x, y) \in \mathcal{R}$ in time polynomial in $|x| + |y|$. (2) For every n, $\mathcal{R}$ restricted to $\Sigma^{\leq n}$ has at most polynomially (in n) many equivalence classes.

To relate parameterized and unparameterized problems, we employ *cross-compositions.* Consider a language $L \subseteq \Sigma^*$ and a parameterized language $Q \subseteq \Sigma^* \times \mathbb{N}$. Then L *cross-composes* into Q if there is a polynomial equivalence relation $\mathcal{R}$ and an algorithm $\mathcal{A}$, referred to as the *cross-composition*, with: $\mathcal{A}$ takes as input a sequence $x_1, \ldots, x_t \in \Sigma^*$ of strings that are equivalent with respect to $\mathcal{R}$, runs in time polynomial in $\Sigma_{i=1}^{t} |x_i|$, and outputs an instance (y, k) of Q such that $k \leq p(\max_{i \in [1..t]} |x_i| + \log(t))$ for a polynomial p. Moreover, $(y, k) \in Q$ if and only if there is an $i \in [1..t]$ such that $x_i \in L$. Cross-compositions are the key to lower bounds for kernels:

▶ **Theorem 8** ([16]). *Assume that an* NP*-hard language cross-composes into a parameterized language Q. Then Q does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/poly.

To show that $\mathsf{BCS}(cs, m)$ does not admit a polynomial kernel, we cross-compose 3-SAT into $\mathsf{BCS}(cs, m)$. Then Theorem 8 yields the following:

▶ **Theorem 9.** $\mathsf{BCS}(cs, m)$ *does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/poly.

Proof Idea. For the cross-composition, we first need a polynomial equivalence relation $\mathcal{R}$. Assume some standard encoding of 3-SAT-instances over a finite alphabet Γ. We let two encodings φ, ψ be equivalent with respect to $\mathcal{R}$ if both are proper 3-SAT-instances and have the same number of clauses and variables.

Let $\varphi_1, \ldots, \varphi_t$ be instances of 3-SAT that are equivalent with respect to $\mathcal{R}$. Then each φ_i has exactly ℓ clauses and k variables. We can assume that the set of variables is $\{x_1, \ldots, x_k\}$. To handle the evaluation of these, we introduce the NFAs $A_i, i \in [1..k]$, each storing the value of x_i. We further construct an automaton B that picks one out of the t formulas φ_j. Automaton B tries to satisfy φ_j by iterating through the ℓ clauses. To satisfy a clause, B chooses one out of the three variables and requests the corresponding value.

The request by B is synchronized with the memory M. After every such request, M either ensures that the sent variable x_i actually has the requested value or stops the computation. This is achieved by a synchronization with the corresponding variable automaton A_i, which keeps the value of x_i. The number of context switches lies in $\mathcal{O}(\ell)$ and the size of the memory in $\mathcal{O}(k)$. Hence, all conditions for a cross-composition are met. ◀

4 Local Parameterization

In the previous section, we considered a parameterization of BCS that was global in the sense that the threads shared the number of context switches. We now study a parameterization that is *local* in that every thread is given a budget of context switches.

We would like to have a measure for the amount of communication between processes and consider only those computations in which heavily interacting processes are scheduled adjacent to each other. The idea relates to [43], where it is observed that a majority of concurrency bugs already occur between a few interacting processes.

Given a word $u \in Ш_{i\in[1..t]} L(A_i)$, we associate with it a graph that reflects the order in which the threads take turns. This *scheduling graph* of u is the directed multigraph $G(u) = (V, E)$ with one node per thread that participates in u, $V \subseteq [1..t]$, and edge weights $E : V \times V \to \mathbb{N}$ defined as follows. Value $E(i, j)$ is the number of times the context switches from thread i to thread j in u. Formally, this is the number of different decompositions $u = u_1.a.b.u_2$ of u so that a is in the alphabet of A_i and b is in the alphabet of A_j. Note that $E(i, i) = 0$ for all $i \in [1..t]$. In the following we refer to directed multigraphs simply as graphs.

In the scheduling graph, the degree of a node corresponds to the number of times the thread has the processor. The *degree* of a node n in $G = (V, E)$ is the maximum over the outdegree and the indegree, $deg(n) = max\{indeg(n), outdeg(n)\}$. As usual, the outdegree of a node n is the number of edges leaving the node, $outdeg(n) = \sum_{n'\in V} E(n, n')$, the indegree is defined similarly. To see the correspondence, observe that a scheduling graph can have three kinds of nodes. The *initial node* is the only node where the indegree equals the outdegree minus 1, and the thread has the processor outdegree many times. For the *final node*, the outdegree equals the indegree minus 1, and the thread computes for indegree many contexts. For all other (usual) nodes, indegree and outdegree coincide. Any scheduling graph either has one initial, one final, and only usual nodes or, if the computation starts and ends in the same thread, only consists of usual nodes. The degree of the graph is the maximum among the node degrees, $deg(G) = max\{deg(n) \mid n \in V\}$.

Our goal is to measure the complexity of schedules. Intuitively, a schedule is simple if the threads take turns following some pattern, say round robin where they are scheduled in a cyclic way. To formalize the idea of scheduling patterns, we iteratively contract scheduling graphs to a single node and measure the degrees of the intermediary graphs. If always the same threads follow each other, we will be able to merge the nodes of such neighboring threads without increasing the degree of the resulting graph. This discussion leads to a notion of scheduling dimension that we define in the following paragraph. In the full version of the paper [14], we elaborate on the relation to an established measure: The carving-width.

Given a graph $G = (V, E)$, two nodes $n_1, n_2 \in V$, and $n \notin V$, we define the operation of *contracting* n_1 *and* n_2 into the fresh node n by adding up the incoming and outgoing edges. Formally, the graph $G[n_1, n_2 \mapsto n] = (V', E')$ contains the vertices $V' = (V \setminus \{n_1, n_2\}) \cup \{n\}$ and has the edge weights $E'(n', n) = E(n', n_1) + E(n', n_2)$, $E'(n, n') = E(n_1, n') + E(n_2, n')$, and $E'(m, m') = E(m, m')$ for all other nodes. Using iterated contraction, we can reduce a graph to only one node. Formally, a *contraction process* of G is a sequence $\pi = G_1, \dots, G_{|V|}$ of graphs, where $G_1 = G$, $G_{k+1} = G_k[n_1, n_2 \mapsto n]$ for some $n_1, n_2 \in V(G_k)$ and $n \notin V(G_k)$, $k \in [1..|V|-1]$, and $G_{|V|}$ consists of a single node. The degree of a contraction process is the maximum of the degrees of the graphs in that process, $deg(\pi) = max\{deg(G_i) \mid i \in [1..|V|]\}$. The *scheduling dimension* of G is $sdim(G) = min\{deg(\pi) \mid \pi$ a contraction process of $G\}$.

We study the complexity of BCS when parameterized by the scheduling dimension. To this end, we define the language of all words where the scheduling dimension is bounded by the parameter $sdim \in \mathbb{N}$: $SDL(\Sigma, t, sdim) = \{u \in (\Sigma \times [1..t])^* \mid sdim(G(u)) \leq sdim\}$.

> *Bounded Context Switching – Local Parameterization* (BCS-L)
> **Input:** $S = (\Sigma, M, (A_i)_{i\in[1..t]})$ and bound $sdim \in \mathbb{N}$ on the scheduling dimension.
> **Question:** Is $L(S) \cap SDL(\Sigma, t, sdim) \neq \emptyset$?

▶ **Theorem 10.** BCS-L *can be solved in time* $\mathcal{O}^*((2m)^{4sdim}4^t)$.

We present a fixed-point iteration that mimics the definition of contraction processes by iteratively joining the interface sequences of neighboring threads. Towards the definition of a suitable composition operation, let the *product* of two interface sequences σ and τ be $\sigma \otimes \tau = \bigcup_{\rho\in\sigma Ш\tau} \rho\downarrow$. Language $\rho\downarrow$ consists of all interface sequences ρ' obtained by summarizing subsequences in ρ. Summarizing $(r_1, r'_1)\dots(r_n, r'_n)$ where $r'_1 = r_2$ up to $r'_{n-1} = r_n$ means to contract the sequence to (r_1, r'_n). We write $\sigma \otimes^k \tau$ for the variant of the product that only returns interface sequences of length at most $k \geq 1$, $(\sigma \otimes \tau) \cap (Q \times Q)^{\leq k}$.

Our algorithm computes a fixed point over the powerset lattice (ordered by inclusion) $\mathcal{P}(\ (Q\times Q)^{\leq sdim} \times \mathcal{P}([1..t])\ )$. The elements are *generalized interface sequences*, pairs consisting of an interface sequence together with the set of threads that has been used to construct it. We generalize $\otimes^k$ to this domain. For the definition, consider (σ_1, T_1) and (σ_2, T_2). If the sets of threads are not disjoint, $T_1 \cap T_2 \neq \emptyset$, the sequences cannot be merged and we obtain $(\sigma_1, T_1) \otimes (\sigma_2, T_2) = \emptyset$. If the sets are disjoint, we define $(\sigma_1, T_1) \otimes^k (\sigma_2, T_2) = (\sigma_1 \otimes^k \sigma_2) \times \{T_1 \cup T_2\}$. The fixed-point iteration is given by $L_1 = \bigcup_{i\in[1..t]} IF(A_i) \times \{\{i\}\}$ and $L_{i+1} = L_i \cup (L_i \otimes^{sdim} L_i)$. The following lemma states that it solves BCS-L. We elaborate on the complexity in the full version of the paper [14].

▶ **Lemma 11.** BCS-L *holds iff the least fixed point contains* $((q_{init}, q_{final}), T)$ *for some* T.

Problem BCS-L can be generalized and can be restricted in natural ways. We discuss both options and show that variants of the above algorithm still apply.

Let BCS-L-ANY be the variant of BCS-L where each thread is given a budget of running cs times, but where we do not make any assumption on the scheduling. Still, the scheduling dimension is bounded by $t \cdot cs$. The above algorithm solves BCS-L-ANY in time $\mathcal{O}^*((2m)^{4t\cdot cs}4^t)$.

Fixing the Scheduling Graph. We consider BCS-L-FIX, a variant of BCS-L where we fix a scheduling graph together with a contraction process of degree bounded by $sdim$. We are interested in finding an accepting computation that switches contexts as depicted by the fixed graph. Formally, BCS-L-FIX takes as input an SMCP $S = (\Sigma, M, (A_i)_{i\in[1..t]})$, a scheduling graph G, and a contraction process π of G of degree at most $sdim$. The task is to find a word $u \in L(S)$ such that $G(u) = G$. Our main observation is that a variant of the above algorithm applies and yields a running time polynomial in t.

▶ **Theorem 12.** BCS-L-FIX *can be solved in time* $\mathcal{O}^*((2m)^{4sdim})$.

Fixing the scheduling graph $G = (V, E)$ and contraction process π has two crucial implications on the above algorithm. First, we need to contract interface sequences according to the structure of G. To this end, we introduce a new product. Secondly, instead of a fixed point we can now compute the required products iteratively along π. Hence, we do not have to maintain the set of threads in the domain but can compute on $\mathcal{P}((Q \times Q)^{\leq sdim})$.

Towards obtaining the algorithm, we first describe the new product that summarizes interface sequences along the graph structure. Let σ and τ be interface sequences. Further, let $\rho \in \sigma Ш \tau$. We call a position in ρ an *out-contraction* if it is of the form $(q, q')(p, p')$ so that (q, q') belongs to σ, (p, p') belongs to τ, and $q' = p$. Similarly, we define *in-contractions*. These are positions where a pair of states of τ is followed by a pair of σ. The *directed product* of σ and τ is defined as: $\sigma \odot_{(i,j)} \tau = \bigcup_{\rho \in \sigma Ш \tau} \rho \downarrow_{(i,j)}$. The language $\rho \downarrow_{(i,j)}$ contains all interface sequences ρ' obtained by summarizing subsequences of ρ, in total containing exactly i out-contractions and j in-contractions. Note that for $\sigma \in (Q \times Q)^n$ and $\tau \in (Q \times Q)^k$, the directed product contracts at $i + j$ positions and yields: $\sigma \odot_{(i,j)} \tau \subseteq (Q \times Q)^{n+k-(i+j)}$.

Now we describe the iteration. First, we may assume that $V = [1..t]$. Otherwise, the non-participating threads in S can be deleted. We distinguishes two cases.

In the first case, we assume that G has a designated initial vertex v_0 and final vertex v_f. Let $\pi = G_1, \dots, G_t$. The iteration starts by assigning to each $v \in V$ the set $S_v = \mathit{IF}(A_v) \cap (Q \times Q)^{deg(v)}$. For S_{v_0}, we further require the first component of the first pair occurring in an interface sequence to be q_{init}. Similarly, for S_{v_f} we require that the second component of the last pair is q_{final}.

Now we iterate along π: For each contraction $G_{j+1} = G_j[n_1, n_2 \mapsto n]$, we compute $S_n = (S_{n_1} \odot_{(i,k)} S_{n_2})$, where $i = E(n_1, n_2)$ and $k = E(n_2, n_1)$. Then $S_n \subseteq (Q \times Q)^{deg(n)}$, where $deg(n)$ is the degree of n in G_{j+1}. Let $V(G_t) = \{w\}$. Then the algorithm terminates after S_w has been computed.

For the second case, suppose that no initial vertex is given. This means that initial and final vertex coincide. Then we iterate through all vertices in V, designate any to be initial (and final), and run the above algorithm. The correctness is shown in the following lemma.

▶ **Lemma 13.** BCS-L-FIX *holds iff* $(q_{init}, q_{final}) \in S_w$.

Round Robin. We consider an application of BCS-L-FIX. We define BCS-L-RR to be the round-robin version of BCS-L. Again, each thread is given cs contexts, but now we schedule the threads in a fixed order: First thread A_1 has the processor, then A_2, followed by A_3 up to A_t. For a new round, the processor is given back to A_1. The computation ends in A_t.

▶ **Proposition 14.** BCS-L-RR *can be solved in time* $\mathcal{O}^*(m^{4cs})$.

The problem BCS-L-RR can be understood as fixing the scheduling graph to a cycle where every node i is connected to $i + 1$ by an edge of weight cs for $i \in [1..t - 1]$ and the nodes t and 1 are connected by an edge of weight $cs - 1$. We can easily describe a contraction process: Contract the vertices 1 and 2, then the result with vertex 3 and up to t. We refer to this as π. Then we have $deg(\pi) = cs$. Hence, we have constructed an instance of BCS-L-FIX.

An application of the algorithm for BCS-L-FIX takes time at most $\mathcal{O}^*(m^{4cs})$ in this case: Let $G_{j+1} = G_j[n_1, n_2 \mapsto n]$ be a contraction in π with $j < t - 1$. Note that $S_{n_1}, S_{n_2} \subseteq (Q \times Q)^{cs}$. We have $E(n_1, n_2) = cs$ and $E(n_2, n_1) = 0$. Hence, the corresponding set S_n is given by $(S_{n_1} \odot_{(cs,0)} S_{n_2}) \subseteq (Q \times Q)^{cs}$. The directed product $\sigma \odot_{(cs,0)} \tau$ can be computed in linear time: Any sequence ρ' in $\sigma \odot_{(cs,0)} \tau$ is obtained from a sequence $\rho \in \sigma Ш \tau$ by summarizing cs many out-contractions. Since σ and τ both have length cs, ρ has to be the sequence where pairs of states of σ and τ alternatively take turns. Hence, $\sigma \odot_{(cs,0)} \tau$ either only consists of ρ' and it is a linear-time procedure to find it, or is empty. For the last contraction $G_t = G_{t-1}[n'_1, n'_2 \mapsto n']$ we have $S_{n'} = (S_{n'_1} \odot_{(cs,cs-1)} S_{n'_2})$. Similar to $\sigma \odot_{(cs,0)} \tau$, one can compute $\sigma \odot_{(cs,cs-1)} \tau$ in linear time. This avoids the cost of the product, the factor 2^{4cs}, in the complexity estimation.

Lower Bound for Round Robin. We prove the optimality of the algorithm for BCS-L-RR by giving a reduction from k × k Clique. This variant of the classical clique problem asks for a clique of size k in a graph whose vertices are the elements of a $k \times k$ matrix. Furthermore, the clique must contain exactly one vertex from each row. The problem was introduced as a part of the framework in [41]. It was shown that the brute-force approach is optimal: $\mathsf{k} \times \mathsf{k}$ Clique cannot be solved in $2^{o(k \log k)}$ time, unless ETH fails. We transport this to BCS-L-RR.

▶ **Lemma 15.** *Assuming* ETH*,* BCS-L-RR *cannot be solved in time* $2^{o(cs \log(m))}$*.*

5 Discussion

Our main motivation was to find bugs in shared-memory concurrent programs. We restricted our analysis to under-approximations and considered behaviors that are bounded in the number of context switches, the memory size, or the scheduling. While this is enough to find bugs, there are cases where we need to check correctness of a program. We shortly outline an FPT upper bound, as well as a matching lower bound for the problem.

The reachability problem on a shared-memory concurrent program in full generality is PSPACE-complete. However, in real-world scenarios, it is often the case that only few (a fixed number of) threads execute in parallel with unbounded interaction. Thus, a first attempt is to parameterize the system by the number of threads t. But this yields a hardness result. Indeed, the problem with t as a parameter is hard for any level of the W-hierarchy.

We suggest a parameterization by the number of threads t and by a, the maximal size of the thread automata A_{id}. We obtain an FPT-algorithm by constructing a product automaton. The complexity is $\mathcal{O}^*(a^t)$. However, there is not much hope for improvement: By a reduction from $\mathsf{k} \times \mathsf{k}$ Clique, we can show that the algorithm is indeed optimal.

References

1 M. F. Atig. Global model checking of ordered multi-pushdown systems. In *FSTTCS*, volume 8 of *LIPIcs*, pages 216–227. Schloss Dagstuhl, 2010.

2 M. F. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan. On bounded reachability analysis of shared memory systems. In *FSTTCS*, volume 29 of *LIPIcs*, pages 611–623. Schloss Dagstuhl, 2014.

3 M. F. Atig, A. Bouajjani, and T. Touili. Analyzing asynchronous programs with preemption. In *FSTTCS*, volume 2 of *LIPIcs*, pages 37–48. Schloss Dagstuhl, 2008.

4 M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, volume 5201 of *LNCS*, pages 356–371. Springer, 2008.

5 M.F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *LMCS*, 7(4), 2011.

6 J. Barnat, L. Brim, I. Cerná, P. Moravec, P. Rockai, and O. Simecek. Divine - A tool for distributed verification. In *CAV*, volume 4144 of *LNCS*, pages 278–281. Springer, 2006.

7 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *STOC*, pages 67–74. ACM, 2007.

8 A. Björklund, P. Kaski, and L. Kowalik. Constrained multilinear detection and generalized graph motifs. *Algorithmica*, 74(2):947–967, 2016.

9 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *JCSS*, 75(8):423–434, 2009.

10 A. Bouajjani, M. Emmi, and G. Parlato. On sequentializing concurrent programs. In *SAS*, volume 6887 of *LNCS*, pages 129–145. Springer, 2011.

11 C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *IWPEC*, volume 5917 of *LNCS*, pages 75–85. Springer, 2009.

12 J.F. Cantin, M.H. Lipasti, and J.E. Smith. The complexity of verifying memory coherence and consistency. *TPDS*, 16(7):663–671, 2005.

13 J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *JCSS*, 72(8):1346–1367, 2006.

14 P. Chini, J. Kolberg, A. Krebs, R. Meyer, and P. Saivasan. On the complexity of bounded context switching. *CoRR*, abs/1609.09728, 2017. URL: `https://arxiv.org/abs/1609.09728`.

15 M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. *ACM TALG*, 12(3):41:1–41:24, 2016.

16 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.

17 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159. IEEE, 2011.

18 S. Demri, F. Laroussinie, and P. Schnoebelen. A parametric analysis of the state explosion problem in model checking. In *STACS*, volume 2285 of *LNCS*, pages 620–631. Springer, 2002.

19 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

20 A. Durand-Gasselin, J. Esparza, P. Ganty, and R. Majumdar. Model checking parameterized asynchronous shared-memory systems. In *CAV*, volume 9206 of *LNCS*, pages 67–84. Springer, 2015.

21 C. Enea and A. Farzan. On atomicity in presence of non-atomic writes. In *TACAS*, volume 9636 of *LNCS*, pages 497–514. Springer, 2016.

22 J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV*, volume 8044 of *LNCS*, pages 124–140. Springer, 2013.

23 J. Esparza, P. Ganty, and T. Poch. Pattern-based verification for multithreaded programs. *ACM TOPLAS*, 36(3):9:1–9:29, 2014.

24 A. Farzan and P. Madhusudan. The complexity of predicting atomicity violations. In *TACAS*, volume 5505 of *LNCS*, pages 155–169. Springer, 2009.

25 H. Fernau, P. Heggernes, and Y. Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *JCSS*, 81(4):747–765, 2015.

26 H. Fernau and A. Krebs. Problems on finite automata and the exponential time hypothesis. In *CIAA*, volume 9705 of *LNCS*, pages 89–100. Springer, 2016.

27 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

28 F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *WG*, volume 3353 of *LNCS*, pages 245–256. Springer, 2004.

29 M. Fortin, A. Muscholl, and I. Walukiewicz. On parametrized verification of asynchronous, shared-memory pushdown systems. *CoRR*, abs/1606.08707, 2016.

30 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *JCSS*, 77(1):91–106, 2011.

31 F. Furbach, R. Meyer, K. Schneider, and M. Senftleben. Memory-model-aware testing: A unified complexity analysis. *ACM TECS*, 14(4):63:1–63:25, 2015.

32 M. Fürer. Faster integer multiplication. *SICOMP*, 39(3):979–1005, 2009.

33 P. B. Gibbons and E. Korach. Testing shared memories. *SICOMP*, 26(4):1208–1244, 1997.

34 M. Hague. Parameterised pushdown systems with non-atomic writes. In *FSTTCS*, volume 13 of *LIPIcs*, pages 457–468. Schloss Dagstuhl, 2011.

35 D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. *Journal of Complexity*, 36:1–30, 2016.

36 R. Impagliazzo and R. Paturi. On the complexity of k-sat. *JCSS*, 62(2):367–375, 2001.

37 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE, 2007.

38 S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.

39 S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

40 A. Lal and T. W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, volume 5123 of *LNCS*, pages 37–51. Springer, 2008.

41 D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *SODA*, pages 760–776. SIAM, 2011.

42 D. Lokshtanov, N. Misra, and S. Saurabh. Kernelization – preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *LNCS*, pages 129–161. Springer, 2012.

43 S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *ASPLOS*, pages 329–339. ACM, 2008.

44 D. Marx. Can you beat treewidth? In *FOCS*, pages 169–179. IEEE, 2007.

45 D. Marx. Can you beat treewidth? *TOC*, 6(1):85–112, 2010.

46 M. Musuvathi and S. Qadeer. Iterative context bounding for systematic testing of multi-threaded programs. In *PLDI*, pages 446–455. ACM, 2007.

47 H. Ponce de León, F. Furbach, K. Heljanko, and R. Meyer. Portability analysis for axiomatic memory models. PORTHOS: one tool for all models. *To appear at SAS*, 2017.

48 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

49 J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, volume 5757 of *LNCS*, pages 566–577. Springer, 2009.

50 T. Wareham. The parameterized complexity of intersection and composition operations on sets of finite-state automata. In *CIAA*, volume 2088 of *LNCS*, pages 302–310. Springer, 2000.

51 C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *TCS*, 26:287–300, 1983.