Parameterization as Abstraction: A Tractable Approach to the Dataflow Analysis of Concurrent Programs

Vineet Kahlon
NEC Labs America, Princeton, NJ 08540, USA.

Abstract

Dataflow analysis for concurrent programs is a problem of critical importance but, unfortunately, also an undecidable one. A key obstacle is to determine precisely how dataflow facts at a location in a given thread could be affected by operations of other threads. This problem, in turn, boils down to pairwise reachability, i.e., given program locations c_1 and c_2 in two threads T_1 and T_2, respectively, determining whether c_1 and c_2 are simultaneously reachable in the presence of constraints imposed by synchronization primitives. Unfortunately, pairwise reachability is undecidable for most synchronization primitives. However, we leverage the surprising result that the closely related problem of parameterized pairwise reachability of c_1 and c_2, i.e., whether for some n_1 and n_2, c_1 and c_2 are simultaneously reachable in the program $T_1^{n_1} \| T_2^{n_2}$ comprised of n_1 copies of T_1 and n_2 copies of T_2, is not only decidable for many primitives, but efficiently so. Although parameterization makes pairwise reachability tractable it may overapproximate the set of pairwise reachable locations and can, therefore, be looked upon as an abstraction technique. Whereas abstract interpretation is used for control and data abstractions, we propose the use of parameterization as an abstraction for concurrency. Leveraging abstract interpretation in conjunction with parameterization allows us to lift two desirable properties of sequential dataflow analysis to the concurrent domain, i.e., precision and scalability.

1 Introduction

Dataflow analysis is an effective and indispensable technique for analyzing large scale real-life sequential programs. For concurrent programs, however, it has proven to be an undecidable problem [17]. This has created a huge gap in terms of the techniques required to meaningfully analyze concurrent programs (which must satisfy the two key criteria of achieving *precision* while ensuring *scalability*) and what the current state-of-the-art offers.

The key obstacle in the dataflow analysis of concurrent programs is to determine for a control location l in a given thread, how the other threads could affect dataflow facts at l. Equivalently, one may view this problem as one of *precisely* delineating *transactions*, i.e., sections of code that can be executed atomically, based on the dataflow analysis being carried out. The various possible interleavings of these atomic sections then determines interference across threads. This question, in turn, boils down to *pairwise reachability*, i.e., whether a given pair of control locations in two different threads are simultaneously reachable. Indeed, in a global state g, a context switch is required at location l of thread T where a shared variable sh is accessed only if starting at g, some other thread currently at location m can reach another location m' with an access to sh that conflicts with l, i.e., l and m' are pairwise reachable from l and m. In that case, we need to consider both interleavings wherein either l or m' is executed first thus requiring a context switch at l.

A simple strategy for dataflow analysis of concurrent program consists of three main steps (i) compute the analysis-specific abstract interpretation of the concurrent program, (ii) delineate the transactions, (iii) compute dataflow facts on the transition graph resulting by taking all necessary interleavings of the transactions. Pushdown systems (PDSs) [1] provide a natural framework for modeling abstractly interpreted threads. A PDS has a finite control part corresponding to the valuation of the variables of the given thread and a stack which provides a means to model recursion. Step (ii) then reduces to pairwise reachability of interacting PDSs in the presence of scheduling constraints imposed by synchronization primitives. While the reachability problem for a single PDS is efficiently decidable, it becomes undecidable for PDSs interacting via standard synchronization primitives like locks, rendezvous (Wait/Notify) and broadcasts (Wait/NotifyAll). This is the key reason for the tractability gap between dataflow analysis of sequential and concurrent programs. In this paper, we propose *parameterization* as a means to bypassing this intractability barrier.

Given control locations c_1 and c_2 of PDSs P_1 and P_2, respectively, c_1 and c_2 are said to be *parameterized pairwise reachable* if for some n_1 and n_2, they are simulta-

neously reachable in the system $P_1^{n_1}||P_2^{n_2}$ comprised of the interleaved parallel composition of n_i copies P_i, i.e., $\exists n_1, n_2 : P_1^{n_1}||P_2^{n_2} \models \mathsf{EF}(c_1 \wedge c_2)$. One would expect reasoning about the system with just two PDSs P_1 and P_2 to be easier than reasoning about an unbounded number of copies of P_1 and P_2. Surprisingly, however, for many important cases of practical interest, parameterized pairwise reachability is not merely tractable but efficiently so. This key insight of delineating transactions based on parameterized pairwise reachability instead of pairwise reachability of control states helps us in surmounting the undecidability barrier for dataflow analysis of concurrent programs.

Clearly, pairwise reachability implies parameterized pairwise reachability but the reverse is not true. Thus parameterized pairwise reachability may over-approximate the set of reachable pairs of control states. However, it is useful to consider this problem for many important reasons. First, for PDSs interacting via primitives like locks, the sets of pairwise and parameterized pairwise reachable control states are the same. Thus for locks we do not lose any reachability information by resorting to parameterization. Secondly, for primitives like rendezvous, even if parameterization may, in general, over-approximate the set of pairwise reachable states, it still allows us to cheaply identify many pairs of control states that are not pairwise reachable and significantly cut down on the set of interleavings that we need to consider for generating dataflow facts. In other words, parameterization can be looked upon as an *abstraction* technique. Whereas abstract interpretation is used for control and data abstractions, we propose the use of parameterization as an added *dimension of abstraction* for handling concurrency. Leveraging abstract interpretation in conjunction with parameterization allows us to lift two desirable properties of sequential dataflow analysis to the concurrent domain, i.e., precision and scalability.

In this paper, we consider concurrent programs comprised of threads communicating with each other using shared variables and synchronizing using standard primitives like locks, wait/notify-style rendezvous and broadcasts. Pairwise reachability in such programs is determined by constraints enforced by (i) synchronization primitives, and (ii) data values. Transaction delineation for such programs is done via an iterative procedure. First, an initial set of transactions is identified by using parameterized pairwise reachability arising only out of synchronization constraints and ignoring shared variables. Clearly, this may over-approximate the set of pairwise reachable states resulting in smaller transactions. Therefore, we next use sound invariants like intervals, octagons and polyhedra [5, 6, 7, 15] to slice away unreachable portions of the given concurrent program. These invariants, which also track shared variables, are computed for the entire concurrent program based on the transactions computed in the previous step. Next,

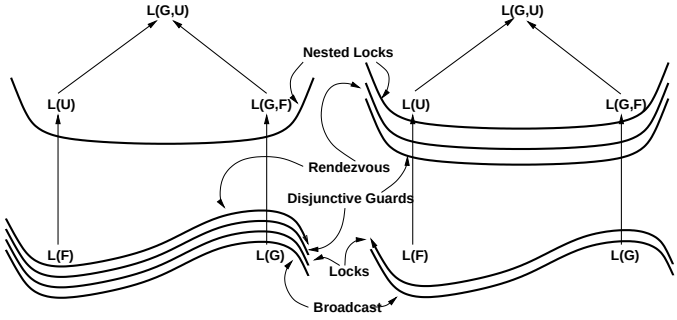

Figure 1. Decidability Boundary: Model Checking vs. PMCP

on this sliced program we again compute the parameterized pairwise reachable control states which are then used to refine transactions computed in the previous step. The whole process is repeated till we can no longer identify larger transactions (see sec 2). Once the transactions have been delineated, dataflow analysis on the product graph of the transactions can be carried out as in the sequential case.

Apart from applications in dataflow analysis for concurrent programs, parameterized reachability, or more broadly, the *Parameterized Model Checking Problem (PMCP)*, is also of independent interest. Indeed, many critical applications like protocols for networking, cache coherence, and multi-core architectures running multi-threaded software, among others, are, by nature, parameterized. As a concrete example, Linux device drivers are supposed to work correctly irrespective of the number of threads executing them. For such applications, the goal is to establish correctness of programs of the form $U_1^{n_1}||...||U_m^{n_m}$ irrespective of the number n_i of threads executing the code for driver U_i. Given a temporal property f, the PMCP is to decide whether for some $n_1, ..., n_m : U_1^{n_1}||...||U_m^{n_m} \models f$. To further appreciate the importance of the PMCP, consider the problem of establishing the absence of data races in a program $U_1||U_2$ comprised of threads U_1 and U_2 running two, possibly distinct, device drivers. Then, if we establish the absence of a data race in the parameterized system $U_1^n||U_2^m$ comprised of arbitrarily many copies of U_1 and U_2 (which can be more tractable), it automatically establishes data race freedom for $U_1||U_2$. This is particularly important since for many applications, like device drivers, even though we might be interested only in the correctness of a system with a fixed number of threads, the underlying notion of correctness is, in fact, parameterized correctness.

In this paper, for each of the standard synchronization primitives, we precisely classify the (linear) temporal logic fragments for which the PMCP is decidable and, in some cases, provide efficient algorithms for the decidable fragments. Correctness properties are expressed using Linear Temporal Logic without the next-time operator X, viz., LTL$\backslash$X. Each formula of LTL$\backslash$X is built using the temporal

operators F "eventually," U "until," G "always," but without X "next-time"; the boolean connectives $\neg, \vee, \wedge$; and atomic propositions interpreted over the local control states of PDSs. Note that absence of the "next-time" operator X makes the logic stuttering insensitive which is usual when reasoning about parameterized systems.

Our new results show that decidability of the PMCP hinges on the set of temporal operators allowed in the correctness property thereby providing a natural way to characterize LTL fragments for which the PMCP is decidable. We use $L(Op_1, ..., Op_k)$, where $Op_i \in \{\mathsf{F}, \mathsf{U}, \mathsf{G}\}$, to denote the fragment comprised of formulae of the form $\mathsf{E}f$, where f is an LTL$\backslash$X formula in positive normal form (PNF), viz., only atomic propositions are negated, built using the temporal operators $Op_1, ..., Op_k$ and the Boolean connectives $\vee$ and $\wedge$. Here E denotes the "existential path quantifier". Obviously, $L(\mathsf{U}, \mathsf{G})$ is the full-blown double-indexed LTL$\backslash$X. Specifically, we show the following (see fig 1: fragments above each curve are undecidable for that primitive).

(a) The PMCP for $L(\mathsf{F}, \mathsf{G})$ and $L(\mathsf{U})$ is, in general, undecidable even for systems wherein the PDSs *do not interact at all* with each other. Thus in order to get decidability of the PMCP for PDSs, interacting or not, we have to restrict ourselves to either the sub-logic L(F) or the sub-logic L(G). For these sub-logics, decidability of the PMCP depends on the synchronization primitive used by the PDSs.

(b) For the sub-logic L(F), we show that the PMCP is efficiently decidable for PDSs interacting via pairwise or asynchronous rendezvous and nested locks but remains undecidable for broadcasts and non-nested locks. The decidability for pairwise rendezvous (and for asynchronous rendezvous) is surprising given the undecidability of model checking systems comprised of two (even isomorphic) PDSs interacting via pairwise rendezvous for reachability – a cornerstone undecidability result for model checking interacting PDSs [17]. Our new results show that the PMCP for PDSs interacting via pairwise rendezvous is not only decidable but efficiently so. This is especially interesting as it illustrates that for rendezvous switching to the parameterized version of the problem makes it more tractable.

(c) For the fragment L(G), we show that the PMCP is decidable for pairwise and asynchronous rendezvous and locks (even non-nested ones). This settles the PMCP for all the standard synchronization primitives.

As applications, we illustrate the usefulness of our techniques in computing range, octagonal and polyhderal invariants for concurrent programs. It is shown that such invariants can, in turn, be used to statically reduce a large fraction of bogus data race warnings resulting from lockset-based methods [13]. Our experiments were conducted on a suite of Linux drivers with known race conditions downloaded from `www.kernel.org`.

```
void Alloc_Page() {
a1:  pt_lock(&plk);
a2:  if(pg_count ≥ LIMIT){
a3:    pt_wait(&pg_lim,&plk);
a4:    pg_count++;
a5:    pt_unlock(&plk);
a6:  } else {
a7:    lock(&count_lock);
a8:    pt_unlock(&plk);
a9:    page = alloc_page();
a10:   if(page)
a11:     pg_count++;
a12    unlock(&count_lock);
     }
}

void Dealloc_Page() {
b1:  pt_lock(&plk);
b2:  if (pg_count==LIMIT){
b3:    pt_send(&pg_lim, &plk);
b4:    pg_count--;
b5:    pt_unlock(&plk);
b6:  } else{
b7:    lock(&count_lock);
b8:    pt_unlock(&plk);
b9:    pg_count--;
b10:   unlock(&count_lock);
     }
}
```

Figure 2. An Example Program

2 Motivation

We consider the problem of static warning generation for data race bugs in concurrent programs. Classical warning generation has three main steps: (i) determine all control locations in each thread with shared variable accesses, (ii) compute the set of locks held at each of these locations, and (iii) each pair of control locations in different threads where (a) the same shared variable is accessed, (b) at least one of the accesses is a write operation, and (c) disjoint locksets are held, is flagged as a data race warning.

The main weakness of lockset-based static warning generation techniques is that too many bogus warnings may be generated. The key reason for this is that such techniques typically ignore conditional statements, and so locations c_1 and c_2 in different thread constituting a data race warning might not be pairwise reachable in the given concurrent program. However, using dataflow analysis such as constant folding and interval, octagon and polyhedral analyses [5, 6, 7, 15] (lifted to concurrent programs), a significant fraction of these bogus warnings can be weeded out as is illustrated by the following example.

Example. Consider the example concurrent program shown in figure 2 wherein different threads may execute the *Page_Alloc* and *Page_Dealloc* functions. The counter *pg_count* stores the number of pages allocated so far. The *Page_Alloc* routine tries to allocate a new page. If the number of pages already allocated has reached the maximum allowed limit *LIMIT*, then it waits (*a3*) until a page is deallocated by the *page_dealloc* routine and a waiting thread signaled (*b3*). For brevity, we have have used *pt_lock, pt_unlock, pt_send, pt_wait* for the Pthread functions *pthread_mutex_lock, pthread_mutex_unlock, pthread_cond_send, pthread_cond_wait*

The shared variable *pg_count* is accessed at locations *a4, a11, b4* and *b9*. The lockset at locations *a4* and *b4* is

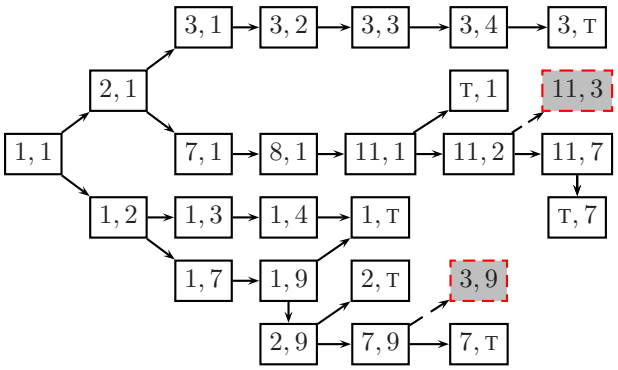

Figure 3. The product transition graph for the example in fig 2.

$\{plk\}$ and at *a11* and *b9* is $\{count\_lock\}$. Since these two locksets are disjoint, the pairs of control locations $(a4, b9)$ and $(a11, b4)$ are labeled as sites where potential data races could arise. However, both *(a4, b9)* and *(a11, b4)* are *bogus*. Importantly, these warnings can be detected as bogus via simple static analyses as we now illustrate.

Figure 3 shows the product graph constructed, on the fly, by the abstract interpreter running an *interval analysis*. An interval over integers is of the $[\ell_i, u_i]$ wherein $\ell_i, u_i \in \mathbf{Z} \cup \pm\infty$ are integers or $\pm\infty$, and $\ell_i \leq u_i$. We denote the empty interval by the symbol $\perp$. Let I denote the set of all intervals over $\mathbf{Z}$. The interval domain consists of assertions of the form $x_i \in [\ell, u]$, associating each variable with an interval containing its possible values.

Each node $\langle i, j\rangle$ denotes the simultaneous reachability of the label ai by the first thread and label bj by the second. Shaded nodes (and their descendants) are shown to be unreachable by our analysis. Consider the edge from the nodes $\langle a11, b2\rangle \rightarrow \langle a11, b3\rangle$. The edge is prohibited because of the invariant $0 \leq \mathsf{pg\_count} < \mathsf{PG\_LIM}$ that is computed at the location $\langle a11, b2\rangle$. However, the edge $b2 \rightarrow b3$ in the code is guarded by the condition $\mathsf{pg\_count} == \mathsf{PG\_LIM}$. Therefore, the invariant proves that the edge is never taken in any execution and therefore, the configuration is never reached. As a result of our analysis, we have established that the nodes $a11$ and $b3$ cannot be simultaneously reached. Together with the lockset analysis, this suffices to show that the program is race-free. *End Example.*

To carry out dataflow analysis, we first need to use pairwise reachability information to delineate the transactions of the given concurrent program. In determining pairwise reachability, we handle (i) synchronization constraints via our new procedures for parameterized reachability, and (ii) data constraints via sound invariants like octagons, etc. Note that unlike synchronization primitives, parameterized pairwise reachability is undecidable for PDSs interacting via shared variables which is why these constraints are handled via sound invariants. First an initial set of (coarse) transactions is identified by using parameterized pairwise reachability based only on synchronization constraints and ignoring shared variables (step 1 of algo. 1). These transactions are then used to compute the initial set of octagonal/polyhedral invariants (step 3). However, based on these sound invariants, it may be possible to prune away unreachable parts of the program. On this sliced program, we again compute (synchronization-based) parameterized pairwise reachable control states which may yield larger transactions (step 4). This, in turn, may lead to sharper invariants. The process of progressively refining transactions by leveraging synchronization constraints and sound invariants in a dovetailed fashion continues till we reach a fix-point.

Algorithm 1 Computing the Product Transaction Graph

1: Construct a product transaction graph G by using only lock and rendezvous constraints.
2: **repeat**
3: Compute range/octagonal/polyhedral invariants and, if possible, prune paths from G resulting in H.
4: Compute a new product transaction graph G' taking into account the pruning of G that resulted in H.
5: **until** G and G' are the same

3 System Model

We consider a family of systems of the form $U_1^{n_1}||...||U_m^{n_m}$ comprised of an arbitrary number n_i of copies of a *template* thread U_i. Each template U_i is modeled as a *Pushdown System (PDS)* [1]. Formally, a PDS is a five-tuple $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$, where P is a finite set of *control states*, *Act* is a finite set of *actions* containing the *empty* action e, Γ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times Act \times (P \times \Gamma^*)$ is a finite set of *transition rules*. If $((p, \gamma), a, (p', w)) \in \Delta$ then we write $\langle p, \gamma\rangle \overset{a}{\hookrightarrow} \langle p', w\rangle$ or $p \overset{a:\gamma\rightarrow w}{\rightarrow} p'$. A *configuration* of $\mathcal{P}$ is a pair $\langle p, w\rangle$, where $p \in P$ denotes the control location and $w \in \Gamma^*$ the *stack content*. We call c_0 the *initial configuration* of $\mathcal{P}$. Let $\mathcal{C}$ be the set of all configurations of $\mathcal{P}$. For each action a, we define a relation $\overset{a}{\rightarrow} \subseteq \mathcal{C} \times \mathcal{C}$ as follows: if $\langle q, \gamma\rangle \overset{a}{\hookrightarrow} \langle q', w\rangle$, then $\langle q, \gamma v\rangle \overset{a}{\rightarrow} \langle q', wv\rangle$ for every $v \in \Gamma^*$. The jth copy of U_i, in $U_1^{n_1}||...||U_m^{n_m}$, denoted by $U_i[j]$, communicates with the other threads via shared variables and synchronizes using standard primitives like locks, pairwise or asynchronous rendezvous, and broadcasts.

1. **Locks.** Locks are standard primitives used to enforce mutually exclusive access to shared resources. We say that a concurrent program accesses locks in a *nested* fashion iff

along each computation of the program, a thread can only release the last lock that it acquired along that computation and that has not yet been released [12]. Nested locks are important as practical programming guidelines used by developers often require that locks be used in a nested fashion. In fact, in Java (version 1.4) and C# locking is syntactically guaranteed to be nested.

2. **Pairwise and Asynchronous Rendezvous.** Rendezvous are motivated by *Wait/Notify* primitives of Java/Pthreads Library. The pairwise rendezvous transitions of a thread are labeled with *send* or *receive* actions of the form $a!$ and $b?$, respectively. A pair of transitions belonging to different threads and labeled with $l!$ and $l?$ are called *matching*. A rendezvous transition $tr_1 : a \xrightarrow{r} b$ of a thread T_i is enabled in global state of a concurrent program, iff there exists a thread T_j other than T_i, in local state c such that there is a matching rendezvous transition of the form $tr_2 : c \xrightarrow{r'} d$. Executing the rendezvous involves both the pairwise send and receive transitions tr_1 and tr_2 firing synchronously as a result of which T_i and T_j transit to b and d, respectively, in one atomic step.

The only difference between pairwise and asynchronous rendezvous is that an asynchronous send transition $a \xrightarrow{l\uparrow} b$ (labeled with $l \uparrow$ instead of $l!$) is non-blocking, viz., can be executed irrespective of whether a matching receive $c \xrightarrow{l\downarrow} d$ (labeled with $l \downarrow$ instead of $l?$) is present or not. A receive transition, on the other hand, does require a matching send to be currently enabled with both the send and receive transitions then being fired atomically.

3. **Broadcast** Broadcast transitions are labeled with send and receive action symbols of the form $a!!$ and $a??$, respectively. A send transition that is enabled can always be fired. A receive transition, on the other hand, can only be fired if there exists an enabled matching broadcast send transition. Broadcasts differ from rendezvous is that executing a broadcast send transition forces not merely one but all other processes with matching receives to fire in one atomic step.

For simplicity, we will formulate our results for parameterized systems with a single template U. For multiple templates we will simply indicate how to carry out the generalization. Given a global computation x of U^n, we use $x[i,j]$ to denote the sequence resulting from projecting x onto the local computation sequence of threads $U[i]$ and $U[j]$.

Correctness Properties. We consider the PMCP for formulae expressed in double-indexed Linear Temporal Logic without the next-time operator (LTL$\backslash$X). A temporal formula f is a *double-indexed* formula over PDSs $U[i]$ and $U[j]$ if each of its atomic propositions may be interpreted over pairs of control states of $U[i]$ and $U[j]$, in which case it is denoted by $f(i,j)$. If the atomic propositions of f are interpreted over the control states of just one process, say $U[i]$, then it is denoted by $f(i)$ and referred to as a single-index formula.

Conventionally, a system $\mathcal{M}$ satisfies a given LTL formula f, denoted by $\mathcal{M} \models f$, if f is satisfied along each path starting at the initial state of $\mathcal{M}$. Using the universal path quantifier A, we may write this as $\mathcal{M} \models \mathsf{A}f$. Equivalently, one can model check for the dual property $\neg\mathsf{A}f = \mathsf{E}\neg f = \mathsf{E}g$, where E is the existential path quantifier. Furthermore, we can assume that g is in *positive normal form (PNF)*, viz., the negations are pushed inwards as far as possible. We use $L(Op_1, ..., Op_k)$, where $Op_i \in \{\mathsf{F}, \mathsf{U}, \mathsf{G}\}$, to denote the fragment of double-indexed LTL$\backslash$X comprised of formulae of the form $\mathsf{E}f$ in positive normal form (only atomic propositions are negated) where f is built using the operators $Op_1, ..., Op_k$ and $\vee$ and $\wedge$.

The Branching-time Logic B(F). We use B(F) to denote the set of branching-time formulae built using the temporal operator AF, the boolean operators $\vee$ and $\wedge$, and atomic propositions. In this paper, we do not deal with the model checking of branching-time logics. B(F) is used only in an intermediate step in the decision procedure for L(G) .

The Parameterized Model Checking Problem. Given finitely many template PDSs $U_1, ..., U_n$ and a temporal property f, the Parameterized Model Checking Problem (PMCP) is to decide whether $\exists n_1, ..., n_k : U^{n_1}||...||U^{n_k} \models f$, where f is double-indexed LTL$\backslash$X formula.

Multi-Automata. Let $\mathcal{P} = (P, Act, \Gamma, \mathbf{c}_0, \Delta)$ be a pushdown system where $P = \{p_1, ..., p_m\}$. A *$\mathcal{P}$-multi-automaton* ($\mathcal{P}$-MA for short) [1] is a tuple $\mathcal{A} = (\Gamma, Q, \delta, I, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Gamma \times Q$ is a set of transitions, $I = \{s_1, ..., s_m\} \subseteq Q$ is a set of initial states and $F \subseteq Q$ is a set of final states. Each initial state s_i corresponds to the control state p_i of $\mathcal{P}$.

We define the transition relation $\longrightarrow \subseteq Q \times \Gamma^* \times Q$ as the smallest relation satisfying the following:

- if $(q, \gamma, q') \in \delta$ then $q \xrightarrow{\gamma} q'$,
- $q \xrightarrow{\epsilon} q$ for every $q \in Q$, and
- if $q \xrightarrow{w} q''$ and $q'' \xrightarrow{\gamma} q'$ then $q \xrightarrow{w\gamma} q'$.

A multi-automaton can be thought of as a *data structure* that is used to succinctly represent (potentially infinite) regular sets of configurations of a given PDS. Towards that end, we say that multi-automaton $\mathcal{A}$ accepts a configuration $\langle p_i, w\rangle$ if $s_i \xrightarrow{w} q$ for some $q \in F$. The set of configurations recognized by $\mathcal{A}$ is denoted by *Conf*($\mathcal{A}$). A set of configurations is *regular* if it is recognized by some MA.

4 Undecidability Barriers

We start with two undecidability results for the PMCP for systems comprised of PDSs that do not even interact with each other.

Theorem 1. *The PMCPs for $L(\mathsf{U})$ and $L(\mathsf{G},\mathsf{F})$ are undecidable for systems comprised of non-interacting PDSs.*

An important consequence of the above results is that for more expressive systems wherein PDSs interact using some synchronization mechanism, we need to focus only on the remaining fragments, viz., $L(\mathsf{F})$ and $L(\mathsf{G})$.

5 Pairwise and Asynchronous Rendezvous

We now present decision procedures for the PMCPs for L(F) and L(G) for PDSs interacting via rendezvous. We start with a provably efficient procedure for computing the set of all *parameterized reachable* control locations of U. This is not only critical for dataflow analysis of concurrent programs (sec. 2) but is also required for formulating decision procedures for L(F) and L(G) .

5.1 Parameterized Reachability

Definition 2 (Parameterized Reachability). *We say that a control state c (configuration* **c***) of template process U is parameterized reachable iff there exists a reachable global state* **s** *of U^n, for some n, with a process in control state c (configuration* **c***).*

We begin by showing that we can pump up the multiplicity of each parameterized reachable configuration of U to any arbitrarily large value. This relieves us of the burden of tracking the multiplicities of configurations of U.

Proposition 3 (Unbounded Multiplicity). *Let R be the set of all parameterized reachable configurations of U and let R' be a finite subset of R. Then given l, for some m, there exists a finite computation of U^m leading to a global state s with at least l copies of each configuration in R'.*

Importantly, the above result allows us to reduce the PMCP for k-wise reachability, viz., $\mathsf{EF}(c_1 \wedge ... \wedge c_k)$ (presence of a data race), to the PMCP for single control state reachability.

Corollary 4. *$\exists n, U^n \models \mathsf{EF}(c_1 \wedge ... \wedge c_k)$ iff for each $i \in [1..k]$, c_i is parameterized reachable.*

The PMCP for Control State Reachability. Parameterized control state reachability for interacting PDSs is complicated by the requirement to satisfy constraints arising both out of (i) synchronization primitives, and (ii) context-free reachability introduced by the stack.

Motivating Example. Consider the template process U shown in figure 4. Suppose that we want to decide whether for some n, $U^n \models \mathsf{EF}c_1$. We start with the set $R_0 = \{c_0\}$ containing only the initial state c_0 of U. We then construct a series of sets R_0,...,R_m, where R_{i+1} is gotten from R_i by adding new control states that become parameterized reachable assuming that all states in R_i are parameterized reachable. In constructing R_{i+1} from R_i, the handling of the

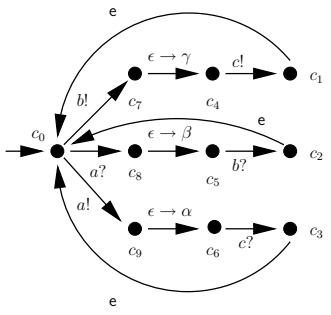

Figure 4. Template U

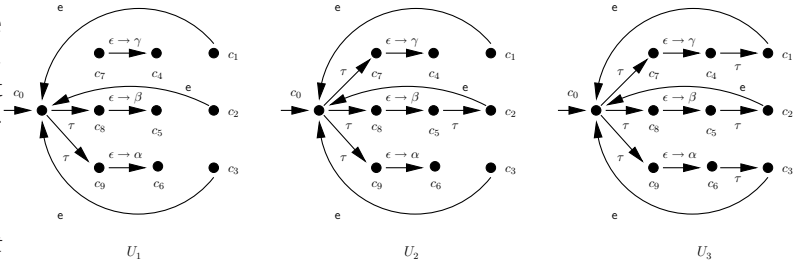

Figure 5. Fixpoint Computation

(i) synchronization, and (ii) context-free reachability constraints is done in a *dovetailed* fashion.

First, in order to satisfy the synchronization constraints, we convert all transitions of the form $a \xrightarrow{p} b$ such that there exists a transition of the form $c \xrightarrow{p'} d$, where p and p' are matching send and receive rendezvous actions with $c \in R_i$, to an internal transition of the form $a \xrightarrow{\tau} b$, where τ is a special internal action symbol. This is motivated by the fact that since c is parameterized reachable we can, via proposition 3, ensure that if a becomes parameterized reachable (now or in some future iteration), then, for some m, there exists a reachable global state of U^m with a process each in local states a and c. Thus if a becomes reachable, the rendezvous transition $a \xrightarrow{p} b$ can always be enabled and executed and can therefore be treated as an internal transition. In this way, by *flooding* all the control states of R_i, we can remove all the synchronization constraints arising out of pairwise send or receive transitions emanating from control states in R_i (step 4 algo 2).

In order to check that the second constraint, viz., context-free reachability, is satisfied, we can now use any procedure for model checking a single PDS, to determine the set R_c^i of those control states of U that are reachable in the individual PDS U_i (step 5 algo. 2). This gives us the set R_c^i of all the context-free reachable states in U_i. If new control states become reachable via removal of some synchronization constraints in the previous step, they are added to R_{i+1}; else we have reached a fixpoint and the procedure terminates.

In our example, R_0 is initialized to $\{c_0\}$. This enables both the transitions $c_0 \xrightarrow{a!} c_9$ and $c_0 \xrightarrow{a?} c_8$ and hence both of them can be converted to internal transitions resulting in

the template U_1 shown in figure 5. In the second iteration, we note that c_5, c_6, c_8 and c_9 are all reachable control states of template U_1 and so $R_1 = \{c_0, c_5, c_6, c_8, c_9\}$. Now, since both c_0 and c_5 are in R_1, the rendezvous transitions $c_5 \xrightarrow{b?} c_2$ and $c_0 \xrightarrow{b!} c_7$ become enabled and can be converted to internal transitions resulting in the template U_2. In U_2, control states c_2, c_4 and c_7 now become reachable and are therefore added to R_2 resulting in $R_3 = \{c_0, c_2, c_4, c_5, c_6, c_7, c_8, c_9\}$. Finally, since both the control states c_4 and $c_6 \in R_3$, the rendezvous transitions $c_6 \xrightarrow{c?} c_3$ and $c_4 \xrightarrow{c!} c_1$ are converted to internal transitions resulting in the template U_3. Since c_1 and c_3 are reachable control locations of U_3, these control locations are now included in R_4 thereby reaching a fixpoint and leading to termination of our procedure. Since $c_1 \in R_4$, we conclude that c_1 is parameterized reachable.

Algorithm 2 Parameterized Control State Reachability for Rendezvous

1: Initialize $i = 0$ and $R_0 = \{c_0\}$, where c_0 is the initial state of U.
2: **repeat**
3: $i = i + 1$
4: Construct PDS U_i by replacing each pairwise send(receive) transition of template U of the form $a \xrightarrow{p} b$, such that there exists a matching receive(send) transition of the form $c \xrightarrow{p'} d$ where $c \in R_{i-1}$, by the internal transition $a \xrightarrow{\tau} b$ and removing the remaining pairwise send or receive rendezvous transitions.
5: Compute the set R_c^i of context-free reachable control locations of (the individual PDS) U_i.
6: Set $R_i = R_{i-1} \cup R_c^i$.
7: **until** $R_i = R_{i-1}$
8: **return** R_i

Theorem 5 *Algorithm 2 returns the set of parameterized reachable control states of U.*

Complexity Analysis. We start by noting that in each iteration of Algorithm 2, we add at least one new control state to R_i. Thus the algorithm terminates in at most $|P|$ times, where P is the set of control states of U. During the ith iteration, we need to decide for each control state in $P \setminus R_i$ whether it is context-free reachable in U_{i+1} which, by using a model checking procedure for PDSs (see [1]), can be accomplished in $O(|U|^3)$ time, where $|U|$ is the size of U. Each step therefore takes at most $O(|U|^4)$ time. Thus this naive analysis shows that algorithm 2 runs in $O(|U|^5)$.

Improving the Complexity. We can, in fact, improve the complexity of the algorithm to $O(|U|^4)$. Instead of recomputing the set of reachable control states from scratch in step 5 of each iteration, we can use results from previous iterations. Towards that end, as in done when model checking a single PDS [1], we maintain the (regular) set C_i of reachable configurations (as opposed to control states) of U in each of the PDSs U_i, as a single multi-automaton $\mathcal{M}$. Note that for each i, the set of transitions of U_i is a subset of those of U_{i+1}. Thus in each iteration, we compute the additional set of configurations of U that have now become reachable given the new transitions that have been added to U_i to compute U_{i+1}. This is accomplished by performing a pre^*-closure of $\mathcal{M}$ for U_{i+1} and takes time $O(|U|^3)$ (see [1]). Since there are at most $|U|$ iterations, the resulting complexity of this improved procedure is $O(|U|^4)$.

Asynchronous Rendezvous. The procedure for deciding the PMCP for PDSs interacting via asynchronous rendezvous, is essentially the same as Algorithm 2. Now, in constructing PDS U_i, step 4 is modified as follows: We replace *each* asynchronous send transition of template U of the form $a \xrightarrow{l\uparrow} b$, with the internal transition $a \xrightarrow{\tau} b$. On the other hand, in order to replace a receive transition of the form $a \xrightarrow{l\downarrow} b$ with the internal transition $a \xrightarrow{\tau} b$, we need to test whether there exists a matching send transition of the form $c \xrightarrow{l\uparrow} d$ with $c \in R_{i-1}$. The remaining receive asynchronous rendezvous transitions are removed. The time complexity of the algorithm remains the same.

Extension to Multiple Templates. To start with, R_0 contains the initial control state of each of the templates $U_1, ..., U_m$. The set R_i now tracks the union of parameterized reachable control states detected up to the ith iteration in any of the templates. Then, in step 4 of algorithm 2, for each $1 \leq j \leq m$, we construct PDS U_{ji} by replacing each rendezvous send/receive transition $a \xrightarrow{p} b$ in template U_j having an enabled matching receive/send transitions of the form $c \xrightarrow{p'} d$ in any of the templates, where $c \in R_{i-1}$, with the internal transition $a \xrightarrow{\tau} b$.

The Model Checking Procedure for L(F). From the given template $U = (P, Act, \Gamma, c_0, \Delta)$, we define the new template $U_R = (P_R, Act, \Gamma, c_0, \Delta_R)$, where P_R is the set of parameterized reachable control states of U and Δ_R is the set of transitions of U between states of P_R with each pairwise rendezvous send or receive transition converted to an internal transition. Then, using a flooding argument resulting from the unbounded multiplicity result, we can show that we can reduce the PMCP for L(F) formulae to the problem of model checking the system U_R^2 comprised of two non-communicating processes. Indeed, if a formula f has a finite computation x of length l, say, as a model, then at most l pairwise send or receive transitions are fired along x. By the unbounded multiplicity lemma, for some m, there exists a computation y leading to a reachable state of U^m, for some m, with at least l copies of each control state of U_R. In a system with U^{m+2} processes, we first let processes

$U_3, ..., U_{m+2}$ execute y to flood all control states of U_R with multiplicity at least l. Then we are guaranteed that in any computation x of $U[1,2]$ of length not more than l, each rendezvous transition can always be fired via synchronization with one of the processes $U_3, ..., U_{m+1}$ and can therefore be treated as internal transitions. Thus we have.

Theorem 6 (Binary Reduction Result). *For any finite computation x of U^n, where $n \geq 2$, there exists a finite computation y of U_R^2 that is stuttering equivalent to $x[1,2]$.*

As an immediate corollary, it follows that if f has a model which is a finite computation of U^m, for some m, then for some k, $U^k \models f$ iff $U_R^2 \models f$. In particular,

Corollary 7. *For any formula f of $L(\mathsf{F})$, for some m, $U^m \models f$ iff $U_R^2 \models f$.*

Since the model checking problem for systems comprised of two *non-interacting* PDSs is known to be efficiently decidable [11], we have

Theorem 8. *The PMCP for $L(\mathsf{F})$ is decidable in polynomial time in the size of U.*

The Model Checking Procedure for L(G). Reasoning about L(G) formulae which, in general, have infinite models is harder than L(F) formulae. In this case, the flooding argument can no longer be used as along an infinite computation x satisfying such a formula, infinitely many rendezvous transitions may be fired. Since every program in the parameterized family U^n has finitely many processes, we cannot pre-flood the control states of U to ensure that every rendezvous transition fired along x has a matching rendezvous transitions. However, we exploit the fact that the dual of an L(G) formula is of the form $\mathsf{A}g$, where g is built using F, the boolean connectives $\wedge$ and $\vee$ and atomic propositions that are control states of U or negations thereof. Such formulae have finite-tree like models which then allows us to leverage the flooding argument. However, note that $\neg(\exists n : U^n \models f)$ iff $\forall n : U^n \models \neg f$. Thus if we resort to the dual of f, the resulting problem is no longer a PMCP.

We next introduce the notion of a *cutoff* that will play a key role in our decision procedure for $L(\mathsf{G})$ formulae.

Definition 9 (Cutoff). *We say that* cut *is a* cutoff *for a temporal logic formula f and a parameterized family defined by a template U iff for $m \geq$ cut, $U^m \models f$ iff $U^{\mathsf{cut}} \models f$.*

The existence of a cutoff for a formula f is useful as it reduces the PMCP for f to a finite number of standard model checking problems for systems with up to the cutoff number of copies of U. Our algorithm for the PMCP for L(G) is then the following:

1. Given an L(G) formula f, we construct a B(F) (see section 3) formula g equivalent to $\neg f$, viz., $U^m \models \neg f$ iff $U^m \models g$.
2. Compute the cutoff cut for g.
3. For each $m \leq$ cut, check if $U^m \models g$.

For step 3, it suffices to check whether for each $m \leq$ cut, $U^m \models f$, where $f = \neg g$ is an L(G) formula. But model checking L(G) formulae for systems with a finite number of PDSs interacting via pairwise or asynchronous rendezvous is known to be decidable [11]. Thus the missing pieces are to show how to (i) construct a B(F) formula g equivalent to f, and (ii) compute cutoffs for B(F) formulae.

From Linear to Branching-Time Step (i) is accomplished via the following result.

Theorem 10. *Given an $L(\mathsf{G})$ formula f', we can construct a B(F) formula f_b equivalent to $f = \neg f'$, i.e., $U^n \models f$ iff $U^n \models f_b$. Furthermore, f_b is of the form $b_1 \wedge \mathsf{AF}(b_2 \wedge \mathsf{AF}(...) \vee b_0 \wedge \mathsf{AF}(...)) \vee b_0 \wedge \mathsf{AF}(...)$, with $b_i = c_i \wedge d_i$, where $c_i(d_i)$ is either* true *or a control state of $U[1](U[2])$*

Cutoffs for B(F) formulae. The final step is to show how to compute cutoffs for B(F) formulae. Towards that end, we first show how to compute cutoffs for L(F) formulae. There are two main reasons for this. First, cutoffs (small model property) for L(F) are of independent interest. Secondly, it is easier to illustrate the key ideas behind cutoff computations for linear than branching-time properties. The cutoff computation for B(F) formulae parallels that for L(F) formulae except that the whole apparatus is lifted to the branching-time spectrum.

Cutoffs for L(F) formulae. We start by showing a linear-time version of theorem 10.

Theorem 11 *Given an $L(\mathsf{F})$ formula f, we can re-write f as $g_0 \vee ... \vee g_k$, with each g_i being of the form $b_0 \wedge \mathsf{F}(b_1 \wedge \mathsf{F}(...))$, where $b_i = c_i \wedge d_i$, where $c_i(d_i)$ is either* true *or a control state of $U[1](U[2])$*

Then the maximum of the cutoff bounds for g_i, where $i \in [0..k]$, gives us a cutoff of f. Finally, the following result allows us to reduce the problem of computing cutoffs for $L(\mathsf{F})$ to computing cutoffs for single-index properties of the form $c_1 \wedge \mathsf{F}(c_2 \wedge \mathsf{F}(...))$.

Proposition 12. *Given a formula $f = c_1 \wedge d_1 \wedge \mathsf{F}(c_2 \wedge d_2 \wedge \mathsf{F}(...))$, the sum of the cutoffs for $f_1 = c_1 \wedge \mathsf{F}(c_2 \wedge \mathsf{F}(...))$ and $f_2 = d_1 \wedge \mathsf{F}(d_2 \wedge \mathsf{F}(...))$ is a cutoff for f.*

Cutoffs for Single-Index Properties. As a final step, we show how to compute cutoffs for single-index properties of the form $f = c_1 \wedge \mathsf{F}(c_2 \wedge \mathsf{F}(...))$. As is usual, we assume, without loss of generality, that each rendezvous send/receive transition tr of U has a unique matching receive/send transition, denoted by $\bar{tr}$. Let x be a finite computation satisfying f. The key idea is to flood every control state of U with enough multiplicity to ensure that every

rendezvous transitions fired along $x[1]$ (f is single-index) is always enabled via synchronization with an enabled transition of a flooded process. Towards that end, for each control state c of U, let Tr_c^f be the set of pairwise rendezvous send/receive transitions of the form $tr : c \rightarrow d$ such that the matching rendezvous $\overline{tr}$ is fired along $x[1]$. Let $n_c^f = \Sigma_{tr \in Tr_c^f} n_{\overline{tr}}^f$, where $n_{\overline{tr}}^f$ is the number of times $\overline{tr}$ was fired along $x[1]$. Then if we flood c with multiplicity n_c^f, we will ensure that each rendezvous transition fired along $x[1]$ that matches a rendezvous emanating from c can always be fired by $U[1]$. Then, in order to estimate a cutoff for f, all we need to do is estimate the number of copies of U required to flood each control state c with multiplicity at least n_c^f. For that, we note that if N_c is a cutoff for $\mathsf{EF}c$ then $cut' = \Sigma_d n_d^f N_d$ processes suffice. Once the flooding has been accomplished, we need one extra process to execute the local computation $x[1]$ satisfying f. Thus

Theorem 13. *$\Sigma_d n_d^f N_d + 1$ is a cutoff for f.*

Cutoff computation, thus reduces to computing upper bounds for N_d and n_d^f which in turn depend on upper bounds for n_{tr}^f – the number of times tr (or, more broadly, a given rendezvous transition) is fired along some path of U comprised of parameterized reachable states satisfying f. Computation of these bounds for PDSs is complicated by context-free reachability introduced by the stack. To handle that, we leverage the notion of a *Weighted Multi-Automaton (WMA)* which is a Multi-Automaton (MA) (sec. 3) with each of its transitions labeled with a non-negative integer.

Definition 14 (Weighted Multi-Automaton). *Given a PDS $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$, a weighted multi-automaton is a tuple $\mathcal{M} = (\Gamma, Q, \delta, w, I, F)$, where $\mathcal{M}' = (\Gamma, Q, \delta, I, F)$ is a multi-automaton for $\mathcal{P}$ and $w : \delta \rightarrow \mathbb{Z}$ is a function mapping each transition of $\mathcal{M}'$ to a non-negative integer.*

The *weight of a finite path* x of $\mathcal{M}$ is defined to be the sum of the weights of all transitions appearing along x. Given states s and t of $\mathcal{M}$, we use $s \overset{u:b}{\rightarrow} t$ to denote the fact that there is path having weight b in $\mathcal{M}$ from s to t and labeled with u. In order to estimate an upper bound for the number of rendezvous transitions fired along a computation satisfying f, we proceed by constructing a WMA $\mathcal{M}_f$ for f which captures the (regular) set of configurations of U satisfying f. Then if b is the weight of an accepting path in $\mathcal{M}_f$, we show that there exists a path of U along which at most b pairwise rendezvous transitions are fired.

Note that $f = c_1 \wedge \mathsf{F}(c_2 \wedge \mathsf{F}(...))$ is built using the operators F and conjunction $\wedge$ between atomic propositions that are control states and formulae of the form $\mathsf{F}g$. Thus in order to construct $\mathcal{M}_f$, it suffices to show how to construct WMAs for $\mathsf{F}g$ and $c \wedge h$ where c is a control state (assuming that we are given WMAs $\mathcal{M}_g$ and $\mathcal{M}_h$ for g and h, respectively). Then given an L(F) formula f, repeated application of these constructions inside out starting with the WMAs for the atomic propositions of f gives us $\mathcal{M}_f$.

Weighted Multi-Automaton for $\mathsf{F}g$. Let $\mathcal{M}_0$ be a given WMA accepting the set of regular configurations of U satisfying g. Starting at $\mathcal{M}_0$, we construct a series of WMAs $\mathcal{M}_0, ..., \mathcal{M}_m$ resulting in the WMA $\mathcal{M}_m$. We recall from the definition of an MA that for each control state c_i of U, there is an initial state s_i of $\mathcal{M}_0$. We denote by $\rightarrow_k$ the transition relation of $\mathcal{M}_k$. Then for every $k \geq 0$, $\mathcal{M}_{k+1}$ is obtained from $\mathcal{M}_k$ by conserving the set of states and adding new transitions as follows: (i) For each internal transition $c_i \rightarrow c_j$, we add the transition $s_i \overset{\epsilon:0}{\rightarrow} s_j$ with weight 0. (ii) For each pairwise rendezvous send or receive transition $c_i \rightarrow c_j$, we add the transition $s_i \overset{\epsilon:1}{\rightarrow} s_j$ with weight 1. (iii) For each stack transition $c_i \overset{\gamma \hookrightarrow u}{\rightarrow} c_j$ of U, if there exists a path x in $\mathcal{M}_k$ from state s_j to t labeled with u, we add the transition $s_i \overset{\gamma:w_u}{\rightarrow} t$, where w_u is the sum of the weights of the transitions occurring along x. Note that if there exist more than one such paths we may take w_u to be the minimum weight over all such paths.

For configurations s and t of U, let $s \Rightarrow_{\leq b} t$ denote the fact that there is path from s to t along which at most b rendezvous transitions are fired. A key result then is

Theorem 15. *If $s_j \overset{w:b}{\rightarrow}_i q$, then $\langle c_j, w \rangle \Rightarrow_{\leq b_1} \langle c_k, v \rangle$ for some c_k and v such that $s_k \overset{v:b_2}{\rightarrow}_0 q$, where $b = b_1 + b_2$. Moreover if q is the initial state s_l then $c_k = c_l$ and $v = \epsilon$. (for each m, s_m is the initial state of $\mathcal{M}_f$ corresponding to control state c_m of U).*

Weighted Automaton for $c \wedge g$. Accomplished via the standard product construction.

The Procedure. Given an L(F) formula f, we first construct a WMA for each atomic proposition of f by constructing an MA for the atomic proposition and setting the weights of all its transitions to 0. Next, we perform the above operations by traversing the formula f inside out starting from the atomic propositions. Let M_f be the resulting WMA. Using the above result, we have

Theorem 16. *Let configuration $\langle q, u \rangle$ of U be accepted by $\mathcal{M}_f$ and let b be the weight of an accepting path of $\mathcal{M}$ starting from q and labeled with u. Then there exists a finite path of U starting from $\langle q, u \rangle$ and satisfying f such that at most b pairwise rendezvous transition are fired along it.*

Computing N_c. Given a control state c of U, we now show how to compute N_c, viz., a cutoff for $\mathsf{EF}c$. Let c be first discovered as parameterized reachable in the ith iteration in Algorithm 2. The computation of N_c is by induction on i. If $i = 0$, viz., c is the initial state of U, $N_c = 1$. Now assume that N_d is known for each $d \in R_i$, where $i > 0$. Let $c \in R_{i+1} \setminus R_i$. Then there is a path of U_{i+1} comprised of

states of R_i leading to c. Using WMA, we can, compute for each rendezvous transition tr, a bound on n_{tr}, the number of times tr is fired along a path of U_{i+1} satisfying $\mathsf{EF}c$. Also, since by the induction hypothesis, we know the value of N_c for each c of R_i, using theorem 13 then gives us a cutoff for $\mathsf{EF}c$, thus completing the induction step.

This completes the cutoff computation for L(F) formulae. The main steps in the cutoff computation for B(F) formulae are the same as for L(F) formulae - the only difference being that everything is now lifted to the branching-time spectrum. We therefore outline only the key details.

Cutoffs for B(F) . For generating cutoffs for a given B(F) formula f, we need to compute an upper bound on the number of rendezvous transitions fired along each path of a finite tree-like model for f. Since the number of rendezvous transitions fired along each path of a tree may, in general, be different, these numbers need to be tracked individually along each path of the tree.

Preliminaries. We start by recalling some standard concepts used in model checking PDSs for branching-time logics. Whereas when reasoning about linear-time formulae sets of reachable configurations are stored as MAs, when reasoning about branching-time formulae they are stored as Alternating Multi-Automata (AMA) [1] - the key difference being that in an AMA each transition could lead to multiple successors instead of just one as in an MA.

Alternating Multi-Automata Let $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$ be a PDS, where $P = \{p_1, ..., p_m\}$. An *alternating $\mathcal{P}$-multi-automaton* is a tuple $\mathcal{A} = (\Gamma, Q, \delta, I, F)$, where Q is a finite set of states, $\delta \subseteq Q \times \Gamma \times 2^Q$ is a set of transitions, $I = \{s_1, ..., s_m\} \subseteq Q$ is a set of initial states and $F \subseteq Q$ is a set of final states. We define the transition relation $\longrightarrow \subseteq Q \times \Gamma^* \times 2^Q$ as the smallest relation satisfying the following: (i) if $(q, \gamma, Q') \in \delta$ then $q \xrightarrow{\gamma} Q'$, (ii) $q \xrightarrow{\epsilon} \{q\}$ for every $q \in Q$, and (iii) if $q \xrightarrow{w} \{q_1, ..., q_n\}$ and for each $1 \leq i \leq n$, $q_i \xrightarrow{\gamma} Q_i$ then $q \xrightarrow{w\gamma} (Q_1 \cup ... \cup Q_n)$.

An AMA $\mathcal{A}$ accepts a configuration $\langle p_i, w\rangle$ if $s_i \xrightarrow{w} Q$ for some $Q \subseteq F$, where s_i is the initial state of $\mathcal{A}$ corresponding to p_i. The set of configurations recognized by $\mathcal{A}$ is denoted by *Conf*($\mathcal{A}$). Given $w \in \Gamma^*$ and $q \in Q$, a *run* of $\mathcal{A}$ over w starting from q is a finite tree whose nodes are labeled by states in Q and whose edges are labeled by symbols in Γ such that the root is labeled by q and the labeling of the other nodes is consistent with δ.

We now recall the standard procedure for model checking PDSs for mu-calculus formulae. Here, the product of the given PDS U with an alternating automaton/tableaux for the given formula f is first constructed. Then the model checking problem reduces to the computation of pre^*-closure of AMAs representing regular sets of configurations of this product. Such products can be modeled as Alternating Pushdown Systems (APDSs).

Alternating Pushdown System (APDS) An APDS is a three-tuple $\mathcal{P} = (P, \Gamma, \Delta)$, where P is a finite set of *control locations*, Γ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times 2^{(P \times \Gamma^*)}$ is a finite set of *transition rules*. For $(p, \gamma, S) \in \Delta$, each *successor set* of the form $\{(p_1, w_1), ..., (p_n, w_n)\} \in S$ denotes a transition of $\mathcal{P}$ which is written as $(p, \gamma) \hookrightarrow \{(p_1, w_1), ..., (p_n, w_n)\}$. If $(p, \gamma) \hookrightarrow \{(p_1, w_1), ..., (p_n, w_n)\}$, then for every $w \in \Gamma^*$ the configuration $\langle p, \gamma w\rangle$ is an *immediate predecessor* of the set $\{\langle p_1, w_1 w\rangle, ..., \langle p_n, w_n w\rangle\}$.

Computing Cutoffs By theorem 10, it follows that we need to consider only B(F) formulae of the form $f = c_1 \wedge d_1 \wedge \mathsf{AF}(c_2 \wedge d_2 \wedge \mathsf{AF}(...) \vee c_0 \wedge d_0 \wedge \mathsf{AF}(...)) \vee c_0 \wedge d_0 \wedge \mathsf{AF}(...)$, where $c_i(d_i)$ is either *true* or a control state of $U[1](U[2])$. Then using reasoning similar to the one used for proposition 12, we can show that it suffices to compute cutoffs for single-index formulae of the form $h = c_1 \wedge \mathsf{AF}(c_2 \wedge \mathsf{AF}(...) \vee c_0 \wedge \mathsf{AF}(...)) \vee c_0 \wedge \mathsf{AF}(...)$.

To compute the cutoff for h, we follow an approach similar to that for L(F) formulae - the only difference being that we use *Weighted Alternating Multi-Automaton (WAMA)*. Note that since models for branching-time properties are computation trees, we need to keep track of the number of pairwise rendezvous fired along each path in such trees. Furthermore, since the number of pairwise rendezvous fired along different branches of a computation tree might be different, each outgoing transition from a state might need to be assigned a different weight. Thus each transition of a WAMA is a member of the set $(Q \times \Gamma) \times 2^{Q \times \mathbb{Z}}$. Formally,

Definition 17. *Given a APDS $\mathcal{P} = (P, \Gamma, \Delta)$, a WAMA is a tuple $\mathcal{M} = (\Gamma, Q, \delta, I, F)$, where $\delta \subseteq (Q \times \Gamma) \times 2^{Q \times \mathbb{Z}}$ and $\mathcal{M}' = (\Gamma, Q, \delta', I, F)$ is an AMA with $\delta' = \{(s, \gamma, \{t_1, ..., t_m\}) | (s, \gamma, \{(t_1, w_1), ..., (t_m, w_m)\}) \in \delta)\}$.*

In order to compute a WAMA $\mathcal{M}_f$ accepting the set of configurations satisfying $f = c_1 \wedge \mathsf{AF}(c_2 \wedge \mathsf{AF}(...) \vee c_0 \wedge \mathsf{AF}(...)) \vee c_0 \wedge \mathsf{AF}(...)$, we need to show how to compute WAMAs for (i) $c \wedge g$, where c is a control state of U, (ii) $g \vee h$, and (iii) $\mathsf{AF}g$, given WAMAs $\mathcal{M}_g$ and $\mathcal{M}_h$ for g and h, respectively.

Constructing a WAMA for $\mathsf{AF}g$. Let APDS $\mathcal{A}$ be the product of U and the tableaux for $\mathsf{AF}g$. Then, starting at the WAMA $\mathcal{M}_0 = \mathcal{M}_g$, we construct a series of WAMAs $\mathcal{M}_0, ..., \mathcal{M}_p$ resulting in the desired WAMA $\mathcal{M}_p$, accepting the set of configurations satisfying $\mathsf{AF}g$. We denote by $\rightarrow_k$ the transition relation of $\mathcal{M}_k$. Then for every $k \geq 0$, $\mathcal{M}_{k+1}$ is obtained from $\mathcal{M}_k$ by conserving the set of states and adding new transitions as follows:

(i) For each transition of $\mathcal{A}$ resulting from the pairwise rendezvous send or receive transition $p_i \xrightarrow{a} p_j$, we add the transition $s_i \xrightarrow{a:1} s_j$ with weight 1.

(ii) For each transition $\langle p_i, \gamma \rangle \hookrightarrow \{\langle p_{i_1}, u_{i_1} \rangle, ..., \langle p_{i_m}, u_{i_m} \rangle\}$ of A, let $s_{i_j} \xrightarrow{u_{i_j}}_k \{t_{i_j 1}, ..., t_{i_j n_j}\}$, where $j \in [1..m]$. Let T_j be the tree, rooted at s_{i_j} and having $t_{i_j 1}, ..., t_{i_j n_j}$ as its leaves, resulting from the run of $\mathcal{M}_k$ on u_{i_j} starting at s_{i_j}. For $l \in [1..n_j]$, let w_{jl} be the sum of the weights of the transitions appearing along the path in T_j from s_{i_j} to $t_{i_j l}$. Then we add the new transition $s_i \xrightarrow{\gamma}_{k+1} \{(t_{i_1 1}, w_{11}), ..., (t_{i_j p}, w_{jp}), ..., (t_{i_m n_m}, w_{m n_m})\}$ to $\mathcal{M}_{k+1}$.

Then, as in the linear case, we can show the following.

Theorem 18 *For configurations $\langle p_i, u \rangle \in Conf(\mathcal{M}_p)$, let w be the sum of the weights of all transitions in a tree resulting from an accepting run for $\langle p_i, u \rangle$. Then there exists a computation tree of U rooted at s_i and satisfying $\mathsf{AF}g$ such that at most w rendezvous transitions occur along its paths.*

The constructions for $c \wedge g$ and $g \vee h$ are the standard product and union constructions for automata and are therefore omitted. This completes the cutoff computation for L(G).

6 Locks

Leveraging a cutoff result from [8], we have that for $n \geq 2, U^n \models f$ iff $U^2 \models f$, where each process $U[i]$ communicates with others only using locks and f is a double-indexed LTL\X formula. This reduces the problem of deciding the PMCP for f to a (standard) model checking problem for a system comprised of only two PDSs interacting via locks. We now consider the two cases of nested and non-nested locks (see section 3) separately.

Nested Locks For a system with two PDSs interacting via nested locks, the model checking problem for systems with two PDSs is known to be efficiently decidable for both fragments of interest, viz., $L(\mathsf{F})$ and $L(\mathsf{G})$ [10].

Non-Nested Locks. For non-nested locks, the undecidability result for $L(\mathsf{F})$ follows immediately from the undecidability of model checking PDSs interacting via locks for $\mathsf{EF}(c_1 \wedge c_2)$ [12].

For $L(\mathsf{G})$, we show that the problem of model checking a system with PDSs interacting via locks for an $L(\mathsf{G})$ formula f can be reduced to model checking an alternate formula f' for two non-interacting PDSs. Given a template U interacting via the locks $l_1, ..., l_k$, we construct a new template V with control states of the form $(c, m_1, ..., m_k)$. The key idea is to store whether a copy of U is currently in possession of lock l_i in bit m_i which is set to 1 or 0 accordingly as U_i is in possession of l_i or not, respectively. Then we can convert V into a non-interacting PDS by removing all locks from V and instead letting each transition of V acquiring/releasing l_i set m_i to $1/0$, respectively. However, removing locks makes control states which were mutually exclusive in U^2 simultaneously reachable in V^2. In order to restore the lock semantics, while model checking for an L(G) property of the form $\mathsf{E}g$, we instead check for the modified L(G) property $\mathsf{E}(g \wedge g')$, where $g' = \mathsf{G}(\wedge_i(\neg m_i^1 \vee \neg m_i^2))$, with atomic proposition m_i^j evaluating to *true* in global state $\mathbf{s}$ iff in the local control state $(c, m_1^j, ..., m_k^j)$ of process V_j in $\mathbf{s}$, $m_i^j = 1$. Note that g' ensures that in the control states of V_1 and V_2, for each i, the m_i-entry corresponding to lock l_i cannot simultaneously be 1 for both $V[1]$ and $V[2]$, viz., $U[1]$ and $U[2]$ cannot both hold the same lock l_i. Then the problem reduces to model checking two non-interacting PDS for L(G) formulae which is known to be decidable [11]. Thus we have

Driver	KLOC	#War	#Con. Red.	Inv. Red.	Time (secs)
pci_gart	0.6	3	1	1	4
jfs_dmap	0.9	32	13	1	52
hugetlb	1.2	4	1	1	0.9
ctrace	1.4	985	58	3	143
autofs	8.3	20	3	2	12
ptrace	15.4	9	1	1	2
raid	17.2	23	13	6	75
tty_io	17.8	6	3	3	11
multicast	26.1	33228	6	6	16

Table 1. Warning Reduction Data

Theorem 19. *The PMCP for L(G) , is efficiently decidable for PDSs interacting via non-nested locks.*

7 Broadcasts

For PDSs communicating via broadcasts, the PMCP is undecidable for even pairwise reachability, viz., $\mathsf{EF}(c_1 \wedge c_2)$, and hence for most interesting temporal properties.

Theorem 20. *The PMCP for $\mathsf{EF}(c_1 \wedge c_2)$, and hence $L(\mathsf{F})$, is undecidable for PDSs interacting via broadcasts.*

8 Experimental Results

We present results for a suite of 10 Linux device drivers with known data races downloaded from `kernel.org`. The results for lockset-based static warning generation and sound-invariant based filtration are presented in table 1. Here, locksets are computed at locations in two different threads where the same shared variable is accessed. If these locksets are disjoint, a data race warning is issued. In some examples (*ipoib_multicast* and *ctrace_comb*), the same control location was reachable in many different thread contexts and so a large number of warnings were generated for the same bug. This problem is addressed by generating only one warning for every pair of control locations (column 4).

Since lockset-based warning generation typically ignores conditional statements, a pair of locations in two different threads that is labeled as a warning might not be simultaneously reachable in any concrete execution of the given program. This results in a bogus warning and is the main weakness of lockset-based static data race detection techniques. To filter out bogus warnings, we need to track reachability constraints arising out of conditional statements which we carry out via computation of octagonal/polyhedral invariants. Our new warning reduction techniques based on these invariants (that leverage parameterization) were then applied on the warnings left after context reduction. Columns 5 and 6 reflects the number of warnings left after applying the invariant-based reductions and the time taken for these reductions. The experimental data clearly illustrates the efficacy of invariant-based reductions in filtering bogus warnings (cols. 4 vs. 6) – especially in the *ctrace, raid* and *plip* examples. In each of the drivers, a warning was generated for the known data race bug.

9 Conclusions and Related Work

Among prior work, [3] attempts to generalize the techniques given in [1] to model check pushdown systems communicating via CCS-style pairwise rendezvous. However since even reachability is undecidable for such a framework, the procedures are not guaranteed to terminate in general but only for certain special cases, some of which the authors identify. For PDSs interacting via rendezvous, over-approximation techniques to achieve termination while performing reachability are considered in [4]. Bounding the number of context switches to ensure decidability of dataflow analysis for concurrent programs has also been explored [14].

The framework of Asynchronous Dynamic Pushdown Networks has been proposed recently [2]. It allows communication via shared variables which makes the model checking problem undecidable. Decidability is ensured by allowing only a bounded number of updates to the shared variables. Another approach that has been explored is to extend the classical procedure-summary based interprocedural dataflow analysis for sequential programs to concurrent programs via the use of transactions [16]. A recent paper [11] delineates the decidability boundary for the model checking problem for systems with finitely many Interacting Pushdown Systems synchronizing via the standard primitives - locks, rendezvous and broadcasts. The dataflow analysis for asynchronous programs wherein threads can fork off other threads but where threads are not allowed to communicate with each other has been explored [9].

In this paper, we have proposed parameterization as a form of abstraction which when used in conjunction with abstract interpretation can provide a tractable framework for dataflow analysis of concurrent programs.

References

[1] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, LNCS 1243, pages 135–150, 1997.

[2] A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *FSTTCS*, 2005.

[3] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *IJFCS*, volume 14(4), pages 551–, 2003.

[4] S. Chaki, E. Clarke, N. Kidd, T.Reps, and T.Touili. Verifying concurrent message-passing c programs with recursive calls. In *TACAS*, 2006.

[5] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to Abstract interpretation, invited paper. In *PLILP '92*, 1992.

[6] Patrick Cousot and Rhadia Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.

[7] Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *POPL*, 1978.

[8] E.A. Emerson and V. Kahlon. Model checking parameterized resource allocation systems. In *TACAS*, 2002.

[9] R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *POPL*, 2007.

[10] V. Kahlon and A. Gupta. An Automata-Theoretic Approach for Model Checking Threads. In *LICS*, 2006.

[11] V. Kahlon and A. Gupta. On the Analysis of Interacting Pushdown Systems. In *POPL*, 2007.

[12] V. Kahlon, F. Ivančić, and A. Gupta. Reasoning about threads communicating via locks. In *CAV*, 2005.

[13] V. Kahlon, Y. Yang, S. Sankaranarayan, and A. Gupta. Fast and Accurate Static Data-Race Detection for Concurrent Programs. In *CAV*, 2007.

[14] A. Lal, T. Touili, N. Kidd, and T. Reps. Interprocedural Analysis of Concurrent Programs Under a Context Bound. In *TACAS*, 2008.

[15] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO II*, 2001.

[16] S. Qadeer, S. K. Rajamani, and J. Rehof. Summarizing procedures in concurrent programs. In *POPL*, 2004.

[17] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. In *ACM TOPLAS*, 2000.

A Undecidability Barriers

Theorem 1 *The PMCPs for L(U) and L(G, F) are undecidable for systems comprised of non-interacting PDSs.*

Proof We show both the undecidability results by reduction from the problem of deciding the disjointness of the context-free languages accepted by two given Pushdown Automata (PDA). Recall that the language accepted by a PDA $P = (Q, Act, \Gamma, c_0, \Delta, F)$, denoted by $L(P)$, is the set of all words $w \in Act^*$ such that there is a valid path of $\mathcal{P}$ labeled with w leading from its initial to a final control state. Given PDA P_1 and P_2, in order to encode the testing of $L(P_1) \cap L(P_2) = \emptyset$, as a PMCP, we construct a PDS U and a formula f of $L(\mathsf{G}, \mathsf{F})$ such that for some n, $U^n \models f$ iff $L(P_1) \cap L(P_2) = \emptyset$. Since testing the disjointness of the context-free languages accepted by two PDSs is undecidable, the undecidability of the PMCP for $L(G, F)$ follows.

Let $P_i = (Q_i, Act, \Gamma_i, \mathbf{c}_i, \Delta_i, F_i)$. Then U is constructed by taking the union of P_1 and P_2. We add a new control state $\mathbf{c}_0$, a new label *init*, and new transitions labeled with *init* from $\mathbf{c}_0$ to both $\mathbf{c}_1$ and $\mathbf{c}_2$. Formally, $U = (Q_1 \cup Q_2, Act \cup \{init\}, \Gamma_1 \cup \Gamma_2, \mathbf{c}_0, \Delta_1 \cup \Delta_2 \cup \{\mathbf{c}_0 \xrightarrow{init} \mathbf{c}_1, \mathbf{c}_0 \xrightarrow{init} \mathbf{c}_2\})$

We first show the undecidability of the PMCP for *L(*G, F*)* formulae. We construct a formula f_I comprised of the "always" operator G such that for some n, U^n satisfies f_I only along those paths where the execution of transitions of P_1 and P_2 labeled with the same symbol a can only execute back-to-back in copies $U[1]$ and $U[2]$ of U, respectively, in U^n. Thus, broadly speaking, the idea is to use $U[i]$ to simulate P_i and couple them together strongly enough.

Towards that end, let $tr_1 : c_1 \xrightarrow{a} d_1$ and $tr_2 : c_2 \xrightarrow{a} d_2$ be a pair of transitions of $\Delta_1 \cup \Delta_2$ belonging to Δ_1 and Δ_2, respectively, both labeled with a. In U, we replace transition tr_i with the new transitions $tr_{i1} : c_i \xrightarrow{\epsilon} \mathsf{c}^a_{i1}$, $tr_{i2} : \mathsf{c}^a_{i1} \xrightarrow{a} \mathsf{c}^a_{i2}$ and $tr_{i3} : \mathsf{c}^a_{i2} \xrightarrow{\epsilon} d_i$, where c^a_{i1} and c^a_{i2} are newly introduced control states and ϵ is the empty symbol. Then to simulate the synchronization of the firing of tr_1 and tr_2, we impose the condition that the transitions tr_{12} and tr_{22} are always fired back-to-back in processes $U[1]$ and $U[2]$ in U^n. This can be ensured by requiring that the formula $f_I = (c^a_{21} \Rightarrow (c_1 \vee c^a_{11} \vee c^a_{12})) \wedge (c^a_{22} \Rightarrow c^a_{12}) \wedge (d_2 \Rightarrow \neg c^a_{11}) \wedge (c^a_{11} \Rightarrow (c_2 \vee \mathsf{c}^a_{21})) \wedge (c^a_{12} \Rightarrow (c^a_{21} \vee c^a_{22})) \wedge (d_1 \Rightarrow \neg c^a_{21})$ is satisfied in each global configuration along a computation. Here atomic propositions c_2, c^a_{21} and c^a_{22} are interpreted over the local control states of $U[2]$ and the atomic propositions c_1, d_1, c^a_{11} and c^a_{12} over the local control states of $U[1]$. This ensures that $U[i]$ executes the transitions of P_i in U. We don't care what the remaining processes execute. Thus along every path of U^n, where $n \geq 2$, satisfying $\mathsf{G}f_I$, transitions of $U[1]$ and $U[2]$ labeled with the same non-empty action symbol are fired back-to-back with $U[1]$'s transition preceding that of $U[2]$. Then $L(P_i) \cap L(P_2) = \emptyset$ iff for some n, $U^n \models f_I \bigwedge f_F$, where $f_F = \vee_{(f_i, f_j) \in F_1 \times F_2}(\mathsf{F}(f_i \wedge f_j))$, ensures that both $U[1]$ and $U[2]$ simultaneously reach final states of P_1 and P_2, respectively. This gives us the undecidability result for $L(\mathsf{G}, \mathsf{F})$.

For showing undecidability of the PMCP for *L(*U*)* formulae, all we need to ensure is that the formula f_I holds only at each state along a path leading to a global configuration with both $U[1]$ and $U[2]$ in final states of P_1 and P_2, respectively. Note that once $U[1]$ and $U[2]$ are both in final states of P_1 and P_2, respectively, we need no longer check that f_I is satisfied. These conditions can be captured by the formula $f_I \mathsf{U} f_F$. Thus, $L(P_1) \cap L(P_2) = \emptyset$ iff for some n, $U^n \models f_I \mathsf{U} f_F$. This gives us the desired undecidability result. ∎

B Pairwise and Asynchronous Rendezvous

Proposition 3. (Unbounded Multiplicity). *Let R be the set of all parameterized reachable configurations of U and let R' be a finite subset of R. Then given l, for some m, there exists a finite computation of U^m leading to a global state s with at least l copies of each configuration in R'.*

Proof Let $\mathbf{c} \in R'$ be a parameterized reachable configuration of U. Then there exists a finite computation path x of U^m, for some m, leading to a state $\mathbf{s}$ with a process in configuration $\mathbf{c}$. To pump up the multiplicity of $\mathbf{c}$ to at least l, we consider the system U^{lm} comprised of lm copies of U. We can then construct a computation y of U^{lm} by first letting processes $U_1, ..., U_m$ execute x while letting the remaining processes stutter. This results in a global state with at least one copy of $\mathbf{c}$. Next, we let processes $U_{m+1}, ..., U_{2m}$ execute the transitions fired along x while letting the other processes stutter. This results in a global state with at least two copies of $\mathbf{c}$. Repeating the above procedure l times, where in the ith step processes $U_{m(i-1)}, ..., U_{mi}$ execute the transitions fired along x, results in a state with at least l copies of $\mathbf{c}$. Finally, carrying out the above procedure for each parameterized reachable configuration in R' yields the desired state. ∎

Corollary 4. *$\exists n, U^n \models \mathsf{EF}(c_1 \wedge ... \wedge c_k)$ iff for each $i \in [1..k]$, c_i is parameterized reachable.*

Proof.

($\Rightarrow$) Let $U^m \models \mathsf{EF}(c_1 \wedge ... \wedge c_k)$. Then there exists a reachable global state of U^m with a PDS in each of the control states $c_1, ..., c_k$. Thus for each i, c_i is parameterized reachable.

($\Leftarrow$) Conversely, for each i, let c_i be parameterized reachable. Then for each i, there exists a reachable configuration $\mathbf{c}_i$ of U^{m_i}, for some m_i, with a PDS in local control state c_i in $\mathbf{c}_i$. Then, by proposition 1, there exists a reachable global state of U^m, for some m, with a PDS in each of the local configurations $\mathbf{c}_i$. Hence $U^m \models \mathsf{EF}(c_1 \wedge ... \wedge c_k)$, thus completing the proof. ■

Theorem 5. *Algorithm 2 returns the set of parameterized reachable control states of U.*

Proof Let $R_0 \subseteq ... \subseteq R_m = R$ be the sequence of control sets generated by Algorithm 2, where R_i is the set of states after the ith iteration of the loop at line 2. We first show that each control state in R is parameterized reachable. The proof is by induction on i. The base case, $i = 0$, is true as the initial state of U is parameterized reachable. For the induction step, we assume that all control states in R_i are parameterized reachable. Let $c \in R_{i+1} \setminus R_i$. Then there exists is a finite path x of U_{i+1} from its initial state c_0 to c. Given x, we now show how to construct a path y of U^m, for some m, leading to a global state with a process in control location c. Let l be the length of x. From the induction hypothesis and proposition 3, we have that for some m', there exists a finite computation z leading to a reachable global state $\mathbf{s}'$ of $U^{m'}$ with at least $l+1$ copies of each control state of R_i. Then we can construct a computation y of $U^{m'+1}$ by first letting processes $U[2], ..., U[m'+1]$ execute the sequence x leading to state $\mathbf{s}''$, say. Then starting at $\mathbf{s}''$, we let process $U[1]$ execute the sequence x except that the firing of an internal transition that result from the replacement of a pairwise send/receiver transition during the execution of Algorithm 2 is now replaced with the original pairwise send or receive. Note that since there are at least $l+1$ copies of each control location of R_i in $\mathbf{s}''$, each such transition is enabled and can therefore be fired resulting in a valid computation of $U^{m'+1}$ leading to a state with $U[1]$ in local state c.

Conversely, let x be a path of U^m leading to a global computation with a process in control state c. The proof is by induction on the length of path x. A path of length zero consists of the initial configuration of U^n with each process in the initial control state c_0 of U. Since c_0 belongs to R_0 and hence R, we are done. Assume now that the result holds for all paths of length i and suppose that $x = x_0...x_{i+1}$ is a path of length $i+1$. Then each control location appearing along $x' = x_0...x_i$ occurs in $R_j \subseteq R$, for some j. Let $U[1]$ be in local state c in x_{i+1}. If control state c occurs in some global configuration along x', by the induction hypothesis $c \in R$ and we are done. Now assume that c does not occur in any configuration along x'. Since each control state occurring along x' is in R_j, all the pairwise rendezvous transitions fired along x are converted to internal transitions in U_{j+1}. If we replace every pairwise send and receive transition fired along $x[1]$ by an internal transition, we get a valid path of U_{i+1} leading from its initial state to control state c. Thus $c \in R_{j+1} \subseteq R$. By symmetry, the same argument holds for the local state of any process $U[r]$ in global state x_{i+1}. This completes the proof. ■

B.1 The Model Checking Procedure for L(F)

Theorem 6. (Binary Reduction Result). *For any finite computation x of U^n, where $n \geq 2$, there exists a finite computation y of U^2_R such that y is stuttering equivalent to $x[1,2]$.*

Proof.

($\Rightarrow$) Given a finite computation x of U^n, where $n \geq 2$, we can construct a sequence y of states of U^2_R by replacing each pairwise rendezvous send or receive transition fired along $x[1,2]$ by an internal transition. Note that each control location occurring along x is parameterized reachable. Thus each pairwise send or receive transition of U fired along x is replaced by an internal transitions in U^R. Thus we see that y is a stuttering of a valid computation of U^2_R.

($\Rightarrow$) Let y be a finite computation of U^2_R. For each control state c, let n_c be the number of firings, along y, of an internal transition of U_R of the form $c \xrightarrow{\tau} d$ that is gotten by replacing a pairwise rendezvous send or receive transition $c \xrightarrow{l} d$ of U in the construction of U_R. We now use a flooding argument. From the unbounded multiplicity result, we have that there exists a reachable global state of s of U^m for some m, with at least n_c copies of control state c. Then we construct a computation x of U^{m+2}, where we first allow processes $U_3, ..., U_{m+2}$ to execute the finite computation leading to s while letting U_1 and U_2 stutter in their initial states. Then we let processes U_1 and U_2 fire the same transitions as were fired along y except that the firing of an internal transition $tr : c \xrightarrow{\tau} d$ of U_R gotten from a rendezvous send/receive $tr' : c \xrightarrow{l} d$ of U is now replaced by the firing of this original rendezvous transition. The matching receive/send transition $tr'' : c' \xrightarrow{l'} d'$ for tr' is fired by one of the auxiliary processes $U_3, ..., U_{m+2}$ in local control state c'. Note that by our construction s has sufficiently many copies of c' to ensure that tr'' is enabled.

Since the relative ordering of the local transitions of $U_R[1]$ and $U_R[2]$ fired along y is the same as the that of the corresponding transitions (with internal transitions of U_R resulting from rendezvous being replaced with the original transitions) of $U[1]$ and $U[2]$ fired along $x[1,2]$, the two sequences are stuttering equivalent. This completes the proof. ■

Corollary 7. *For any formula f of $L(\mathsf{F})$, for some m, $U^m \models f$ iff $U^2_R \models f$.*

Proof Sketch. Follows easily from the above result by noting that L(F) has models that are finite computations. ■

C Computing Cutoffs

C.1 From Linear to Branching-Time

Let f' be an $L(\mathsf{G})$ formula and let $f = \neg f'$. Then f is of the form $\mathsf{A}g$, where g is a doubly-indexed LTL formula built using the operators $\mathsf{F}, \vee$ and $\wedge$ and atomic propositions that are control states of U or negations thereof. We assume, without loss of generality, that all negations in g have been removed by rewriting every expression $\neg c$, where c is a control location of U as the disjunction all the (finitely many) control states of U other than c). We now show how to construct a B(F) formula f_b equivalent to f, viz., $U^n \models f$ iff $U^n \models f_b$. Using the equivalences $\mathsf{F}(h_1 \vee h_2) \equiv \mathsf{F}h_1 \vee \mathsf{F}h_2$ and $h_1 \wedge (h_2 \vee h_3) \equiv h_1 \wedge h_2 \vee h_1 \wedge h_3$, we can drive the F and $\wedge$ operators in g as far down as possible and rewrite g as $g = g_1 \vee ... \vee g_k$, where for each i, g_i is built using F and $\wedge$.

Let g_s be such that for some i, either (i) g_s is g_i, or (ii) there exists a subformula of g_i of the form $\mathsf{F}g_s$. Then g_s is of the form $p \wedge \mathsf{F}g'_{i1} \wedge ... \wedge \mathsf{F}g'_{ik_i}$, where p is a conjunction of control states of $U[1]$ and $U[2]$. We call p the *base expression* of sub-formula g_s. Let B_i be the set of base expressions of g_i or of g_s, where $\mathsf{F}g_s$ a sub-formula of g_i. Define $B_g = \bigcup_i B_i$. Note that the formula $g_s = p \wedge \mathsf{F}g'_{i1} \wedge ... \wedge \mathsf{F}g'_{ik_i}$ imposes an implicit ordering on the base expressions of g_s. Indeed, in g_s we can define p to be less than the base expressions occurring in each of the sub-formulae $g'_{i1}, ..., g'_{ik_i}$. Intuitively, this can be justified by the fact that along each computation x satisfying g (g_s), there is a state s_p satisfying p such that the states satisfying the base expressions of $g'_{i1}, ..., g'_{ik_i}$ occur after s_p along x.

In fact, with the formula g, we can associate a set BT_g of base expression tuples of the form $(b_1, ..., b_l)$, where each $b_i \in B_g$, reflecting the order in which states satisfying the tuples must appear along a computation satisfying g, viz., the order $b_1, ..., b_l$. Clearly $BT_g = BT_{g_1} \cup ... \cup BT_{g_k}$. To compute BT_{g_i} we build the syntax tree T_i for g_i. Each path of T_i defines an ordering on the base expressions occurring along it. However since g_i is built using the operators F and $\wedge$, along each computation path satisfying g_i only the local ordering among base expressions imposed by each path of T_i (the order in which the base expression occur from the root of T_i to the leaf associated with the path) must be satisfied. Thus in order to compute BT_{g_i} we form tuples that result by interleaving sub-tuples for all paths of T_i in all possible ways while preserving the local orders imposed on the base expressions by different paths of T_i. This captures all the different ways in which the sub-tuples can be reconciled, viz., all possible total orders that can be imposed on the union of all base expression of g.

We say that a computation x of U^n satisfies a tuple $(b_1, ..., b_l)$ of base expressions iff $x \models b_1 \wedge \mathsf{F}(b_1 \wedge \mathsf{F}(b_2 \wedge ...\mathsf{F}(b_l)...))$. Then the following result is immediate

Theorem 10'. *$U^n \models \mathsf{A}g$ iff each computation of U^n satisfies at least one of the base expression tuples occurring in BT_g.*

As a final step, we show that if $BT_g = \{t_1, ..., t_p\}$, we can construct a $B(\mathsf{F})$ formula $f_b(t_1, ..., t_p)$ such that $U^n \models f_b(t_1, ..., t_p)$ iff each computation of U^n satisfies at least one of the base expression tuples occurring in BT_g. The construction proceeds by induction on the cardinality of BT_g.

Base Case: $|BT_g| = 1$. Let $BT_g = \{(b_1, ..., b_l)\}$. Then it is obvious that the formula $f_b = b_1 \wedge \mathsf{AF}(b_1 \wedge (\mathsf{AF}...\mathsf{AF}(b_l)...))$ suffices.

Induction Step: Now assume that $|BT_f| = k > 1$ and $BT_f = \{t_1, ..., t_k\}$. Let $f' = f_b(t_1, ..., t_{k-1})$ and let $t_k = (t_{k1}, ..., t_{kl})$. Then the formula $f_b(t_1, ..., t_k) = t_{k1} \wedge \mathsf{AF}(t_{k2} \wedge \mathsf{AF}(...) \vee f') \vee f'$ suffices. The intuition behind the construction is the following. The formula $t_{k1} \wedge \mathsf{AF}(t_{k2} \wedge \mathsf{AF}(...))$ is satisfied at the initial state iff each path starting at the initial state satisfies the tuple $(t_{k1}, ..., t_{kl})$. However, any path x that does not satisfy the tuple $(t_{k1}, ..., t_{kl})$ must satisfy one of the tuples $t_1, ..., t_{k-1}$, or in other words the formula f'. This leads to the various disjunctions with f' to take into account the cases where only a certain prefix but not the entire tuple $(t_{k1}, ..., t_{kl})$ may be satisfied.

C.2 Cutoffs for L(F) formulae

Theorem 11. *Given an $L(\mathsf{F})$ formula f, we can re-write f in the form $g_0 \vee ... \vee g_k$, with each g_i being of the form $b_0 \wedge \mathsf{F}(b_1 \wedge \mathsf{F}(...))$, where $b_i = c_i \wedge d_i$, where $c_i(d_i)$ is either* true *or a control state of $U[1](U[2])$*

Proof Sketch. Without loss of generality, each atomic proposition of f can be assumed to be of the form c or $\neg c$, where c is control location of U. Rewriting $\neg c$ as the disjunction all the (finitely many) control states of U other than c we can remove all negations from f. Let $f = \mathsf{E}g$. Then, by driving the $\vee$ operator in g as far up as possible we can write $g = g_1 \vee ... \vee g_k$, where for each i, g_i does not contain the $\vee$ operator.

Note that with each $g_l = g(1, 2)$, we can associate a set Seq of finite sequences of ordered pairs of the form (c_i, d_j), where $c_i(d_i)$ is either *true* or a control state of $U[1](U[2]$ resp.) occurring in $g(1, 2)$, that capture all possible orders in which global states satisfying $c_i \wedge d_i$ can

appear along computation paths satisfying $g(1,2)$. For example, with the formula $c_0^1 \wedge \mathsf{F}(c_1^1 \wedge c_4^2) \wedge \mathsf{F}c_3^2$, where c_i^j is true if $U[j]$ is currently in local control state c_i, we can associate the sequences. $(c_0^1, true), (true, c_3^2), (c_1^1, c_4^2)$ and $(c_0^1, true), (c_1^1, c_4^2), (true, c_3^2)$. Thus $U^n \models \mathsf{E}g(1,2)$ iff there exist a sequence $\pi : (c_1, d_1), ..., (c_k, d_k)$ in Seq and a computation path x along which there exists global states satisfying $c_1 \wedge d_1, ..., c_k \wedge d_k$ in the order listed, viz., $x \models f_\pi = c_1 \wedge d_1 \wedge \mathsf{F}(c_2 \wedge d_2 \wedge \mathsf{F}(...))$. ■

Proposition 12. *Given a formula $f = c_1 \wedge d_1 \wedge \mathsf{F}(c_2 \wedge d_2 \wedge \mathsf{F}(... \wedge \mathsf{F}(c_k \wedge d_k)...))$, the sum of the cutoffs for $f_1 = c_1 \wedge \mathsf{F}(c_2 \wedge \mathsf{F}(... \wedge \mathsf{F}(c_k)...))$ and $f_2 = d_1 \wedge \mathsf{F}(d_2 \wedge \mathsf{F}(... \wedge (\mathsf{F}d_k)...))$ is a cutoff for f.*

Proof Sketch. Let cut_i be a cutoff for f_i. Let x and y be computations of U^{cut_1} and U^{cut_2} satisfying f_1 and f_2, respectively. Then there exist global states $x_{i_1}, ..., x_{i_k}$ such that (i) $i_1 \leq ... \leq i_k$, and (ii) either c_j is *true* or the local state of $U[1]$ in x_{i_j} is c_j. Similarly, let $y_{m_1}, ..., y_{m_k}$ be global states occurring along y such that (i) $m_1 \leq ... \leq m_k$, and (ii) either d_j is *true* or $U[2]$ has local state d_j in y_{m_j}. We now consider the system $U^{\mathsf{cut}_1 + \mathsf{cut}_2}$. Clearly its initial state satisfies $c_1 \wedge d_1$. Then we let processes $U[1], U[3], ..., U[\mathsf{cut}_1 + 1]$ execute the sequence of local transitions fired by the processes $U[1], U[2], ..., U[\mathsf{cut}_1]$, respectively, along the subsequence $x_1, ..., x_{i_2}$ resulting in global state $\mathbf{t}_1$, say. Note that the local state of $U[1]$ in $\mathbf{t}_1$ satisfies c_2. Next we let processes $U[2], U[\mathsf{cut}_1 + 2], ..., U[\mathsf{cut}_1 + \mathsf{cut}_2]$ execute the local computations executed by processes $U[2], U[1], U[3], ..., U[\mathsf{cut}_2]$, respectively, along $y_{i_0}, ..., y_{i_1}$. Let $\mathbf{s}_1$ be the resulting state. Note that in $\mathbf{s}_1$, the local states of $U[1]$ and $U[2]$, satisfy c_2 and d_2 respectively, viz., $\mathbf{s}_2 \models c_2 \wedge d_2$. Repeating the process k times where in the jth step we first let processes $U[1], U[3], ..., U[\mathsf{cut}_1 + 1]$ execute the sequence of local transitions fired by the processes $U[1], U[2], ..., U[\mathsf{cut}_1]$, respectively, along the subsequence $x_{i_j}, ..., x_{i_{j+1}}$ and then let processes $U[2], U[\mathsf{cut}_1 + 2], ..., U[\mathsf{cut}_1 + \mathsf{cut}_2]$ execute the local computation executed by processes $U[2]$, $U[1]$, $U[3], ..., U[\mathsf{cut}_2]$, respectively, along $y_{m_j}, ..., y_{m_{j+1}}$ resulting in global state $\mathbf{s}_j$ satisfying $c_j \wedge d_j$. This proves our result ■

Theorem 13. *$\Sigma_d n_d^f N_d + 1$ is a cutoff for f.*

Proof Sketch. Once each control state c is flooded with multiplicity n_d^f using processes $U_2, ..., U_{k+1}$, process U_1 can execute $x[1]$ wherein each rendezvous transition fired by U_1 synchronizes with one of the processes $U_2, ..., U_{k+1}$.

All we need to show now is that $k \leq \Sigma_d n_d^f N_d$. Since N_c is a cutoff of $\mathsf{EF}c$, there exists a computation x_c of U^{N_c} leading to a global state with a process in local state c. In order to get a global state with at least n_c copies of c, we let processes $U[1], ..., U[N_c]$ of $U^{n_c N_c}$ execute x_c to reach a global state $\mathbf{s}_1$ with at least one copy of c. Next, starting at $\mathbf{s}_1$, we let processes $U[N_c + 1], ..., U[N_c + n_c]$ execute x_c to reach a global state $\mathbf{s}_2$ with at least two copies of c. Repeating this processes n_c times results in a global state $\mathbf{s}_{n_c}$ with at least n_c copies of c. Repeating this process for each control state d, then gives us the desired result. ■

C.3 WMAs

Weighted Automaton for $c \wedge g$. Let $\mathcal{M}_1 = (\Gamma, Q_1, \delta_1, w_1, I_1, F_1)$ and $\mathcal{M}_2 = (\Gamma, Q_2, \delta_2, w_2, I_2, F_2)$ be two WMAs where $\mathcal{M}_1$ accepts the set of configurations $\langle c, w \rangle$, $w \in \Gamma^*$, and $\mathcal{M}_2$ the set of configurations satisfying g. Note that each transition of $\mathcal{M}_1$ for atomic proposition c has weight 0. Recall that, by definition of an MA, for each control state p_i of U there is an initial state s_i of $\mathcal{M}$, and vice versa. Thus $I_1 = \{s_1, ..., s_k\} = I_2$, where s_i is the (unique) initial state corresponding to the control state p_i of U. Then the we can construct a WMA $\mathcal{M}$ accepting the intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$ via the standard product construction $\mathcal{M} = (\Gamma, Q_1 \times Q_2, \delta, w, I_1 \times I_2, F_1 \times F_2)$, where $(t_1, t_2) \xrightarrow{l:w} (t_3, t_4) \in \delta$ iff $(t_1 \xrightarrow{l:0} t_3) \in \delta_1$ and $(t_2 \xrightarrow{l:w} t_4) \in \delta_2$. The state $(s_i, s_i) \in Q_1 \times Q_2$ is renamed as s_i in order to ensure a one-to-one correspondence between the control states of U and initial states of $\mathcal{M}$. Essentially, from the set of formulae satisfying g the above construction filters out configurations that are not in control state c.

Theorem 15. *If $s_j \xrightarrow{w:b}_i q$, then $\langle p_j, w \rangle \Rightarrow_{\leq b_1} \langle p_k, v \rangle$, for some p_k and v such that $s_k \xrightarrow{v:b_2}_0 q$, where $b = b_1 + b_2$. Moreover if q is the initial state s_l then $p_k = p_l$ and $v = \epsilon$.*

Proof The proof is by induction on i. The base case, $i = 0$, holds by letting $p_k = p_j$ and $v = w$. Also, we note that if q is an initial state then since there is no transition of $\mathcal{M}_0$ leading into an initial state, we have $w = \epsilon$ and $q = s_j$.

For the induction step, we assume that $i \geq 1$. Let x be a path of $\mathcal{M}_i$ labeled with w and weight b leading from s_j to q. We double induct on the number of transitions of $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$ fired along x. For that we write x as $s_j \xrightarrow{w_1:b_{11}}_i s_{k_0} \xrightarrow{\gamma:b_{12}}_i q' \xrightarrow{w_2:b_{13}}_i q$, where $w = w_1 \gamma w_2$, transition $s_{k_0} \xrightarrow{\gamma:b_{12}}_i q$ belongs to $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$ and $b = b_{11} + b_{12} + b_{13}$. From $s_j \xrightarrow{w_1:b_{11}}_i s_{k_0}$ and the fact that s_{k_0} is an initial state, by applying the induction hypothesis, we have that $\langle p_j, w_1 \rangle \Rightarrow_{\leq b_{11}} \langle p_{k_0}, \epsilon \rangle$.

There are three cases to consider. First, we assume that the addition of the transition $s_{k_0} \xrightarrow{\gamma:b_{12}}_i q'$ results from the stack transition $p_{k_0} \xrightarrow{\gamma \hookrightarrow v_1} p_{k_1}$ in case (iii) of the algorithm

computing the WMA for $\mathsf{F}g$. Then, $s_{k_0} \overset{\gamma:b_{12}}{\rightarrow}_i q'$ was included in $\mathcal{A}_i$ because $s_{k_1} \overset{v_1:b_{12}}{\rightarrow}_{i-1} q'$. Using the induction hypothesis (on i), we have that $\langle p_{k_1}, v_1\rangle \Rightarrow_{\leq b_{21}} \langle p_{k_2}, v_2\rangle$, for some p_{k_2} and v_2 such that $s_{k_2} \overset{v_2:b_{22}}{\rightarrow}_0 q'$, where $b_{12} = b_{21} + b_{22}$. Thus there is a run y of $\mathcal{M}_i$ satisfying $s_{k_2} \overset{v_2:b_{22}}{\rightarrow} q' \overset{w_2:b_{13}}{\rightarrow} q$. Since y has lesser transitions of $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$ fired along it than along x, applying the induction hypothesis we obtain $\langle p_{k_2}, v_2w_2\rangle \Rightarrow_{\leq b_{31}} \langle p_k, v\rangle$, for some p_k and v such that $s_k \overset{v:b_{32}}{\rightarrow}_0 q$, where $b_{22} + b_{13} = b_{31} + b_{32}$. Thus $\langle p_j, w_1\gamma w_2\rangle \Rightarrow_{\leq b_{11}+b_{21}+b_{31}} \langle p_k, v\rangle$ for some p_k and v such that $s_k \overset{v}{\rightarrow}_0 q$. Note that $(b_{11} + b_{21} + b_{31}) + b_{32} = b_{11} + b_{21} + b_{22} + b_{13} = b_{11} + b_{12} + b_{13} = b$

Now consider the case where $s_{k_0} \overset{\epsilon:b_{12}}{\rightarrow}_i q'$ corresponds to the pairwise rendezvous $p_{k_0} \overset{\epsilon}{\rightarrow} p_{k_1}$. Then, $b_{12} = 1$ and $q' = s_{k_1}$. Thus there exists a path $s_{k_1} \overset{w_2:b_{13}}{\rightarrow}_i q$ that has lesser transitions of $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$ fired along it than along x. Thus by the induction hypothesis, we have that $\langle p_{k_1}, w_2\rangle \Rightarrow_{\leq b_{21}} \langle p_k, v\rangle$, for some p_k and v such that $s_k \overset{v:b_{22}}{\rightarrow}_0 q$, where $b_{13} = b_{21} + b_{22}$. Hence $\langle p_j, w_1w_2\rangle \Rightarrow_{\leq b_{12}+1+b_{21}} \langle p_k, v\rangle$ for some p_k and v such that $s_k \overset{v}{\rightarrow}_0 q$. Note that $b_{12}+1+b_{21}+b_{22} = b_{12}+1+b_{13} = b$

The case where $s_{k_0} \overset{\epsilon:b_2}{\rightarrow}_i q'$ corresponds to an internal transition is handled similarly except that now the weight of $s_{k_0} \overset{\epsilon:b_2}{\rightarrow}_i q'$ is 0 instead of 1.

This completes the induction step and proves the result. ∎

D Broadcasts

Theorem 20. *The PMCP for $\mathsf{EF}(c_1 \wedge c_2)$, and hence $L(\mathsf{F})$, is undecidable for PDSs interacting via broadcasts is undecidable.*

Proof. The result follows by reduction from the problem of deciding the disjointness of context-free languages accepted by two given Pushdown Automata (PDA) P_1 and P_2. Recall that the language accepted by a PDA $P = (Q, Act, \Gamma, c_0, \Delta, F)$, denoted by $L(P)$, is the set of all words $w \in \Gamma^*$ such that there is a valid path of P labeled with w leading from the initial to a final control state of P. In order to encode the testing of $L(P_1) \cap L(P_2) = \emptyset$ as a PMCP for pairwise reachability, we proceed as follows:

1. We construct a template U such that the structure of U guarantees that in a system U^n, exactly one copy of U, say $U[1]$, executes the transitions of P_1 and exactly one other copy of U, say $U[2]$, executes the transitions of P_2, with the remaining processes stuck in a newly introduced dead-end state.

2. We make sure that all executions of transitions of $U[1]$ and $U[2]$ labeled with the same action symbol a are matched, viz., the execution of the transition of $U[2]$ immediately follows the execution of $U[1]$. Then testing whether $L(P_1) \cap L(P_2) = \emptyset$, reduces to deciding whether for some n, there exists a reachable global state of U^n with both $U[1]$ and $U[2]$ in final local states, viz., whether $\mathsf{EF}(f_1 \wedge f_2)$ holds for the parameterized system $\{U\}^n$, for some pair of final control states f_1 and f_2 of P_1 and P_2, respectively. The undecidability of the PMCP for $\mathsf{EF}(f_1 \wedge f_2)$ then follows immediately.

We now briefly discuss how to construct the template U described above. For each i, let PDA P_i be the tuple $(Q_i, \Sigma, \Gamma_i, c_i, \Delta_i, F_i)$, where Σ is the common set of action symbols labeling transitions of both P_1 and P_2. Suppose that $a \in \Sigma$ is a non-empty action symbol. Let P_2^b be the PDS that we get from P_2 by replacing each transition $tr_2 : c \overset{a}{\rightarrow} d$ labeled with a, with the pair of broadcast send and receive transitions $tr_{21} : c \overset{a!!}{\longrightarrow} in_{tr?}$ and $in_{tr?} \overset{a??}{\longrightarrow} d$, where $in_{tr?}$ is a newly introduced *intermediate* state. On the other hand, let P_1^b be the PDS that we get from P_1 by replacing each transition $tr_1 : c \overset{a}{\rightarrow} d$ labeled with a, with the pair of broadcast receive and send transitions $tr_{11} : c \overset{a??}{\longrightarrow} in_{tr!}$ and $tr_{12} : in_{tr!} \overset{a!!}{\longrightarrow} d$, where $in_{tr!}$ is a newly introduce intermediate state. Note that the send and receive broadcast transitions are added in P_2^b in the reverse direction relative to each other from that in P_1^b which is crucial in ensuring that requirement 2 is satisfied. Furthermore, in each P_j^b, we introduce from each control state c which does not already have an out going broadcast send transition labeled with $a??$, the new transition from $c \overset{a??}{\rightarrow} \mathsf{end}_j$ of P_j^b, where end_j is a newly introduced dead-end control state of P_j^b. This is done to ensure that in U^n at most two processes execute any *meaningful* computation, the rest eventually ending up in these dead end states. The template U can now be defined by essentially taking the *union* of the two PDSs P_1^b and P_2^b by adding a new initial state i_U; the broadcast send transition $init_1 : \mathsf{i}_U \overset{pick!!}{\rightarrow} c_1$; and the broadcast receive transition $init_2 : \mathsf{i}_U \overset{pick??}{\rightarrow} c_2$. Then it can be shown that $L_1 \cap L_2$ iff for some $n, U^n \models \mathsf{EF}(f_1 \wedge f_2)$, where f_1 and f_2 are final control locations of P_1 and P_2, respectively. The undecidability result then follows as an immediate corollary.