Queue-Dispatch Asynchronous Systems

Gilles Geeraerts   Alexander Heußner
Jean-François Raskin
Université Libre de Bruxelles – Belgium

17$^{\text{th}}$ April, 2012

Abstract

To make the development of efficient multi-core applications easier, libraries, such as Grand Central Dispatch, have been proposed. When using such a library, the programmer writes so-called *blocks*, which are chunks of codes, and dispatches them, using *synchronous* or *asynchronous* calls, to several types of waiting queues. A scheduler is then responsible for dispatching those blocks on the available cores. Blocks can synchronize via a global memory. In this paper, we propose Queue-Dispatch Asynchronous Systems as a mathematical model that faithfully formalizes the synchronization mechanisms and the behavior of the scheduler in those systems. We study in detail their relationships to classical formalisms such as pushdown systems, Petri nets, fifo systems, and counter systems. Our main technical contributions are precise worst-case complexity results for the Parikh coverability problem for several subclasses of our model.

1 Introduction

Over the last decades, the computing power delivered by processors has followed an exponential growing rate, mainly because of a steady increase in the CPU clock rates. Nowadays, however, further increasing the clock rates of CPUs incurs major engineering challenges, and multi-core CPUs have been introduced as an alternative way to achieve more computing power. The introduction of multicore CPUs, however, yields a major paradigm shift for programmers, as they must now write applications *with parallelism in mind*, in order to take advantage of those new platforms. Unfortunately, writing *correct* and *efficient* parallel code is a notoriously difficult task. The actual behaviour of a parallel program stems from the intricate interaction of the threads that run concurrently, and is thus hard to predict, hard to understand, hard to test, and hard to reproduce for debugging. Moreover, some parameters that can have a major influence on the efficiency of the code (such as the number of available cores, or the energy management policy of the platform) are usually known at runtime only. Thus, *automatic* tools to *reason* about such parallel applications running on multi-core platforms (and, in particular, to *verify* them) are sorely needed.

1

In order to help the programmer exploiting parallelism, several high level programming interfaces have been made available. A popular example is *Grand Central Dispatch*, GCD for short, a technology that is present in Mac OS X (since 10.6), iOS (since version 4), and FreeBSD. In GCD, the programmer writes so-called *blocks* which are chunks of codes, and *dispatches* them to waiting *queues*, together with several dependency constraints between those blocks (for instance, one block cannot start before the previous one in the queue has finished). The scheduler is then responsible for *scheduling* those blocks on the available cores, taking into account the current state of the computing platform (number of available cores, and so forth).

So far, to the best of our knowledge, no formal model has been proposed for systems relying on GCD or similar technologies, making those programs *de facto* out of reach of current verification methods and tools. This is particularly unfortunate as the control structure of such programs is very rich. For instance, as it is possible to dispatch an *unbounded* number of blocks to the queues, the state space of such programs is infinite even when the types of the variables are abstracted to finite domains.

Contributions In this paper, we introduce *Queue-Dispatch Asynchronous Systems*, QDAS for short, as a formal model for programs written using libraries such as GCD. Our model is composed of *blocks* (finite transition systems with finite data-domain variables) that can do *asynchronous* (non-blocking) and *synchronous* (blocking) calls to other blocks (recursion is allowed[1]). However, a call does not immediately trigger the execution of the callee: the block is inserted into a fifo queue that can be either *concurrent* or *serial*. In concurrent queues, several blocks can be taken from the queue and executed in parallel, while in serial queues, a block can be dequeued only if the previous block in the queue has completed its execution.

In the following, we formalize the verification question for QDAS through the *Parikh coverability problem*. The Parikh image of a QDAS configuration is a *counting abstraction* of the configuration that counts how many blocks are in each possible control state, see for example [11]. The Parikh coverability problem then asks whether the QDAS can reach a configuration c whose Parikh image *covers* a given Parikh image c', i.e. whether there are, in c, at least as many blocks as in c' in each control state. It is well-known that meaningful verification problems, such as *mutual exclusion* ('are there at least two block in a critical section at the same time ?') can be reduced to this problem.

It is well-known that fifo systems, which are finite automata extended by a single *unbounded fifo queue*, can simulate a Turing machine. Thus one could expect that all non trivial decision problems (in particular, Parikh coverability) on QDAS with at least one (unbounded) queue will be undecidable, independent of the type of queue. While the Parikh coverability is, on the general QDAS model, undecidable, we show in this paper that some natural restrictions of the model enjoy decidability results *even while retaining unbounded fifo queues*. Intuitively, this rather surprising result stems

---

[1]As the reader will note later, classical (*blocking*) procedure calls (and so recursion) can be naturally coded in our model by synchronous calls to blocks. We could have added function calls (and so recursion) explicitly into the syntax of our model however at the cost of an additional unnecessary complexity to our definitions (and without any gain in term of expressiveness).

| Parikh coverability | | queue types | |
	concurrent	serial	both
dispatch synchronous	EXPTIME-C	PSPACE-C	EXPTIME-C
asynchronous	EXPSPACE-C	↯	↯
both	↯	↯	↯

Table 1: Summary of Results, where ↯ means "undecidable"

from the fact that, in QDAS, queues contain *blocks* and not *messages* as in fifo systems. Moreover, running blocks cannot precisely control the *dequeueing*, as this is the responsibility of the scheduler. This prevents some non-trivial and meaningful classes of QDAS from simulating precisely fifo systems and allows to regain decidability.

More precisely, we show that QDAS with *only* synchronous calls are essentially equivalent to pushdown systems with finite domain data-variables, and we show that the Parikh coverability problem is EXPTIME-C for that class (Theorem 1). Second, for synchronous QDAS with only serial queues, the problem is PSPACE-C (Theorem 2). Third, we show that QDAS with *only* asynchronous calls and *only* concurrent queues are essentially equivalent to lossy Petri nets and show that the Parikh coverability problem is EXPSPACE-C for that class (Theorem 3). This decidability border is precise as we show that if we allow either (i) asynchronous calls with synchronous queues, or (ii) synchronous and asynchronous calls with concurrent queues, then the Parikh coverability problem becomes undecidable (Theorem 4 and Theorem 5).
Remark: Due to the lack of space, detailed formal proofs are deferred to the appendix.

Related Works The basic model-checking result for asynchronous programs is the EXPSPACE-hardness for the control-state reachability problem obtained by making formal a link with *multi-set pushdown systems* (MPDS) [14, 8, 12]. These programs however only allow to store pending asynchronous calls in (a set of) multisets and to execute one task per multiset. The close relation between asynchronous programs and Petri nets can also be used to prove additional decidability results [10, 9]. Multi-set pushdown automata are subsumed by *well-structured transition systems with auxiliary storage* and inherit their decidability results presented in [6, 7]. A series of parallel programming libraries and techniques is formalized in [3] with the help of *recursively parallel programs*. These allow to model fork/join based parallel computations based on a reduction to recursive vector addition systems with states. Recursively parallel programs only cover the classical asynchronous models presented above and not the advanced scheduling strategies for different queues that are part of our QDAS.

2 Preliminaries

Grand Central Dispatch (GCD) is a technology developed by Apple [1, 2] that is publicly available at `http://libdispatch.macosforge.org/` under a free license. GCD is the main inspiration for the formal model of queue-dispatch asynchronous systems. In the following, we often present our examples as pseudo code using a syntax inspired by GCD. In the GCD framework, the programmer has to split

```
1  global int[l][m] matrix1, int[m][n] matrix2, int[l][n] matrix
2  global c_queue workqueue
3  global s_queue semaphore
4  global int count

6  block increase():
7    count = count + 1

9  block one_cell(int i, int j):
10   for k in range(m):
11     matrix[i][j]+= matrix1[i][k] * matrix2[k][j]
12   dispatch_s(semaphore,increase())

14 def main():
15   count = 0
16   for i in range(l):
17     for j in range(m):
18       dispatch_a(workqueue,one_cell(i,j))
19   wait(count = l*m)
20   // print the result
```

Figure 1: GCD (-like) program for parallel matrix multiplication

his code into *blocks*. During the execution of a GCD program, one or several *tasks* run
in parallel, each executing a given block (initially, only the `main` block is running).
Tasks can call (or *dispatch* in the GCD vocabulary) other blocks, either *synchronously*
(the call is blocking), or *asynchronously* (the call is not blocking). A *dispatch* con-
sists in inserting the block into a fifo *queue*. In our examples, we use the keywords
dispatch_a and dispatch_s to refer to asynchronous and synchronous dispatches re-
spectively. At any time, the scheduler can decide to *dequeue* blocks from the queues
and to assign them to tasks for execution. All queues ensure that the blocks are de-
queued in FIFO order, however the actual scheduling policy depends on the type of
queue. GCD supports two types of queues: *concurrent queues* allow several tasks from
the same queue to run in parallel, whereas *serial queues* guarantee that *at most one* task
is running per queue. In our examples, concurrent (resp. serial) queues are declared as
global variables of type c_queue (s_queue). In addition, all blocks have access to
the same set of *global variables* (in this work, we assume that the variables range over
finite domains).

Example 1 *Let us consider the pseudo code in Fig. 1 that computes the product of two
integer matrices* matrix1 *and* matrix2 *in a matrix* matrix*. The* main *task forks
a series of* one_cell *blocks. Each* one_cell *computes the value of a single cell of
the result. The parallelism is achieved via the GCD scheduler, thanks to* asynchronous
dispatches *on the* concurrent *queue* workqueue*. Asynchronous dispatches ensures
that* main *is not blocked after each dispatch, and a concurrent queue allow all the*
one_cell *blocks to run in parallel. The variable* count *is incremented each time
the computation of a cell is finished and acts as a semaphore for the* main *block, to
ensure that* matrix *contains the completed result when* main *terminates. As only
reading and writing to a variable are atomic, we need to guarantee exclusive access to*
count *(line 7). This is achieved by a dedicated block* increase *that is dispatched
to the* serial *queue* semaphore*. As only* increase *blocks can increase* count*, this
queue implicitly locks the access to the variable.*

Basic Notations: Given a set S, let $|S|$ denote its cardinality. For an I-indexed family of sets $(S_i)_{i \in I}$, we write elements of $\prod_{i \in I} S_i$ in bold face, i.e., $\mathbf{s} \in \prod_{i \in I} S_i$. The i-component of $\mathbf{s}$ is written $s_i \in S_i$, and we identify $\mathbf{s}$ with the indexed family of elements $(s_i)_{i \in I}$. We use $\uplus$ to denote the disjoint union of sets. An *alphabet* Σ is a finite set of *letters*. We write Σ^* for the set of all *finite words*, over Σ and denote the empty word by ε. The concatenation of two words w, w' is represented by $w \cdot w'$. For a letter $\sigma \in \Sigma$ and a word $w \in \Sigma^*$, let $|w|_\sigma$ be the number of occurrences of σ in w. We use standard complexity classes, e.g., polynomial time (PTIME) or deterministic exponential time (EXPTIME), and mark completeness by appending "-C" (e.g. PSPACE-C).

A **labeled transition system** (LTS) is a tuple $\mathcal{TS} = \langle S, s^0, \Sigma, \Rightarrow \rangle$ where S is a set of *states* (sometimes called *configurations*), $s^0 \in S$ is the initial state, Σ is an alphabet of *actions*, and $\Rightarrow \subseteq S \times \Sigma \times S$ is a (labeled) *transition relation*. We usually write $s \overset{a}{\Longrightarrow} s'$ in place of $(s, a, s') \in \Rightarrow$. A *run* ρ of $\mathcal{TS}$ is a finite alternating sequence $s_0 a_1 s_1 a_2 \ldots a_n s_n$ of configurations and actions where $(s_i, a_{i+1}, s_{i+1}) \in \Longrightarrow$ for all $0 \le i < n$ and $s_0 = s^0$. A configuration s is *reachable* in $\mathcal{TS}$ iff there exists a run $s_0 a_1 s_1 a_2 \ldots a_n s_n$ of $\mathcal{A}$ such that $s_n = s$. We denote by $Reach(\mathcal{TS})$ the set of all reachable configurations of $\mathcal{S}$. Given an LTS $\mathcal{TS}$ and a corresponding state s, the LTS *reachability problem* asks whether $s \in Reach(\mathcal{TS})$.

A **pushdown system** (PDS) is a tuple $\langle X, x^0, \Phi, \Sigma, \Delta \rangle$ where X is a finite set of (control) states, $x^0 \in X$ is the initial state, Φ is the stack alphabet and $\Delta \subseteq X \times \Sigma \times X$ is the set of transitions with $\Sigma = (\{\texttt{push}, \texttt{pop}\} \times \Phi) \cup \{\varepsilon\}$. The semantics of a pushdown system is given as infinite transition system over configurations $S = X \times \Phi^*$ where the stack content is represented by a word in Φ^* that is accessed in *lifo* manner. It is folklore, that a pushdown system with an additional action $\texttt{empty?} \in \Sigma$, that can only be fired if the stack is empty, can be (linearly) reduced to one without. The reachability question for PDS is PTIME-C [4].

Let $\mathbb{D}$ be a finite **data domain** with an *initial element* $d_0 \in \mathbb{D}$, and let $\mathcal{X}$ be a finite set of variables ranging over $\mathbb{D}$. A *valuation* of the variables in $\mathcal{X}$ is a function $\mathbf{d} : \mathcal{X} \to \mathbb{D}$. An *atom* is an expression of the form $x = d$ or $x \neq d$, where $x \in \mathcal{X}$ and $d \in \mathbb{D}$. A *guard* is a finite conjunction of atoms. An assignment is an expression of the form $x \leftarrow v$, where $x \in \mathcal{X}$ and $v \in \mathbb{D}$. Let $\mathsf{guards}\,(\mathcal{X})$, $\mathsf{assign}\,(\mathcal{X})$ and $\mathsf{vals}\,(\mathcal{X})$ denote respectively the sets of all guards, assignments and valuations over variables from $\mathcal{X}$. Guards, atoms and valuations have their usual semantics: for all valuations $\mathbf{d}$ of $\mathcal{X}$ and all $g \in \mathsf{guards}\,(\mathcal{X})$, we write $\mathbf{d} \models g$ iff $\mathbf{d}$ satisfies g.

A **pushdown system with data** is a pushdown system equipped with a finite set of variables $\mathcal{X}$ over a finite domain $\mathbb{D}$. A configuration of a PDS with data is a pair $(s, w, \mathbf{d})$ where s is a control state, w is the stack content, and $\mathbf{d}$ is a valuation of the variables

Proposition 1 *The reachability problem is* EXPTIME-C *for* PDS *with data.*

A **Petri net** (PN) is a tuple $N = \langle P, T, m_0 \rangle$ where P is a finite set of places, a *marking* of the places is function $m : P \to \mathbb{N}$ that associates, to each place $p \in P$ a number $m(p)$ of tokens, T is finite set of transitions, each transition $t \in T$ is a pair (I_t, O_t)
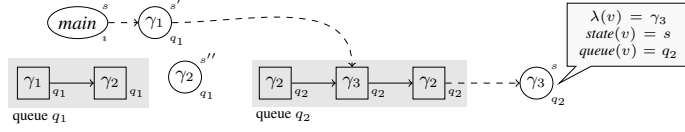
Figure 2: CTG for a QDAS with a concurrent queue q_1 and a serial queue q_2

where $I_t : P \to \{0,1\}$ and $O_t : P \to \{0,1\}$ are respectively the *input* and *output* *functions* of t, and m_0 is the *initial marking*. The preorder $\preceq$ compares markings: $m \preceq m'$ iff $m(p) \leq m'(p)$ for all $p \in P$. As usual, a transition t is enabled in a marking m if $I(t) \preceq m$. When fired from m, t consumes one token in each input place and produces one token in the output places defining a new marking m', this is noted $m \to m'$. This semantics can be formalized using an infinite transition system with set of configurations $\mathbb{N}^P$. A *run* in this transition system is a finite sequence $m_0 m_1 \ldots m_n$ s.t. for all $1 \leq i \leq n$: $m_{i-1} \to m_i$. For a PN N, we denote by *Reach*(N) (resp. *Cover*(N)) the *reachability (coverability) set* of N, i.e. the set of all markings m s.t. there exists a run $m_0 m_1 \ldots m_n$ of N with $m = m_n$ ($m_n \preceq m$). The *coverability problem* asks, given a PN N and a marking m, whether $m \in$ *Cover*(N). It is EXPSPACE-complete [8].

A **fifo system** is a tuple $\langle X, x^0, M, \Delta \rangle$ where X is a finite set of (control) states, $x^0 \in X$ is the initial state, M is the message alphabet and $\Delta \subseteq X \times \Sigma \times X$ is the set of transition rules for $\Sigma = (\{!, ?\} \times M) \cup \{\varepsilon\}$. The semantics of a fifo system is given as infinite transition system over configurations $S = X \times M^*$ where the queue content is represented by a word in M^* that is accessed in a *fifo* manner. It is well known that the control state reachability problem is undecidable for fifo systems [5].

A **two counter system** (2CS) is a tuple $\{X, x^0, \Sigma, \Delta\}$ where X is a finite set of (control) state, $x^0 \in X$ is the initial state, $\Sigma = \{\texttt{incr}, \texttt{decr}, \texttt{is\_zero}\} \times \{1, 2\}$ is the set of actions, and $\Delta \subseteq X \times \Sigma \times X$ is the set of transition rules. The actions $\texttt{incr}(i)$ and $\texttt{decr}(i)$ are interpreted as incrementing and decrementing a counter variable i (for $i = 1, 2$) that has values in $\mathbb{N}$, and testing whether the current counter i equals zero. The semantics of a 2CS is represented by an infinite LTS with states $S = X \times \mathbb{N} \times \mathbb{N}$. The reachability problem for two counter systems (with zero test) is undecidable [13].

3 Queue-dispatch asynchronous systems

Syntax: We are now ready to define our model for GCD-like programs. Let $\mathbb{D}$ be a finite data domain with *initial value* d_0. A *queue-dispatch asynchronous system* (QDAS) $\mathcal{A}$ is a tuple $\langle CQID, SQID, \Gamma, main, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ where:

- $CQID$ and $SQID$ are respectively finite sets of *(c)oncurrent* and *(s)erial queues*;
- Γ is the finite set of *blocks* and $main \in \Gamma$ is the *initial block*.
- $\mathcal{X}$ is a finite set of $\mathbb{D}$-valued variables;
- Σ is the set of *actions*, with $\Sigma = (\{\texttt{dispatch}_\texttt{s}, \texttt{dispatch}_\texttt{a}\} \times (CQID \cup SQID) \times \Gamma \setminus \{main\}) \cup \texttt{guards}(\mathcal{X}) \cup \texttt{assign}(\mathcal{X})$.

- for each $\gamma \in \Gamma$: $\mathcal{TS}_\gamma$ is a tuple $\langle S_\gamma, s^0_\gamma, f_\gamma, \Sigma, \Delta_\gamma \rangle$ where $\langle S_\gamma, s^0_\gamma, \Sigma, \Delta_\gamma \rangle$ is an LTS and $f_\gamma \in S$ a distinct final state.

We assume that $SQID, CQID, \Gamma, \mathcal{X}$, and all S_γ for $\gamma \in \Gamma$ are disjoint from each other. We let $S = \biguplus_{\gamma \in \Gamma} S_\gamma$, $F = \biguplus_{\gamma \in \Gamma} \{f_\gamma\}$, $\Delta = \biguplus_{\gamma \in \Gamma} \Delta_\gamma$, and $QID = SQID \uplus CQID \uplus \{\bot\}$ (where $\bot \notin SQID \cup CQID$). We further assume that $\varepsilon \notin \Sigma$.

Call-task graphs: We formalize the semantics of QDAS using the notion of *call-task graph* (CTG) to describe the system's global configurations. A configuration of a QDAS (see Fig. 2 for an example) contains a set of running tasks, represented by *task vertices* (depicted by round nodes) and a set of called but unscheduled blocks, represented by *call vertices* (square nodes). Call vertices are held by queues, and the linear order of each queue is represented by *queue edges* (solid edges). Synchronous calls add an additional dependency (the caller is waiting for the termination of the callee) that is represented by a *wait edge* (dashed edges) between the caller and the callee. Wait edges are also inserted between the head of a *serial* queue and the running task that has been extracted from this queue (if it exists) to indicate that the task has to terminate before a new block can be dequeued. Note that only vertices without outgoing edges can execute a computation step, the others are currently blocked. Each node v is labeled by two pieces of information: a block $\lambda(v) \in \Gamma$, and a queue identifier $queue(v) \in QID$. When v is a call vertex, $\lambda(v)$ is the queue that contains v, otherwise, $\lambda(v)$ is the queue that contained v before it was scheduled. Task vertices are labeled by their current state $state(v)$ (for convenience, we also label call vertices by the initial state of their respective blocks – not shown in the figure).

Example 2 *The CTG in Fig. 2 depicts a configuration of a QDAS with two queues. Queue q_2 is serial (note the outgoing wait edge to the running task) and contains $\gamma_2 \gamma_3 \gamma_2$, and q_1 is parallel with content $\gamma_1 \gamma_2$. There are 4 active tasks, two of them (`main` and the task running γ_1) are blocked. The task running γ_3 has been dequeued from q_2 and is currently at location s.*

Formally, given a QDAS $\mathcal{A} = \langle CQID, SQID, \Gamma, main, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$, a *call-task graph* over $\mathcal{A}$ is a tuple $G_\mathcal{A} = \langle V, E, \lambda, queue, state \rangle$ where: $V = V_C \uplus V_T$ is a finite set of *vertices*, partitioned into a set V_C of *call vertices* and a set V_T of *task vertices*; $E \subseteq V \times V$ is a set of *edges*; $\lambda : V \to \Gamma$ labels each vertex by a block; $queue : V \to QID$ associates each vertex to a queue identifier; and $state : V \to S$ associates each vertex to an LTS state. For each $q \in QID$, let $V_q = \{v \in V \mid queue(v) = q\}$. The set E is partitioned into the set E_W of *wait edges* and the set $E_Q = \bigcup_{q \in QID} E_q$ of *queue edges* where, for each $q \in QID$, $E_q = E \cap (V_q \times V_q)$. A CTG is *empty* iff $V = \emptyset$. A *path* (of length n) in $G_\mathcal{A}$ is a sequence of vertices $v_0, v_1, \ldots, v_n$ s.t. for all $1 \leq i \leq n$: $(v_{i-1}, v_i) \in E$. Such a path is *simple* iff $v_i \neq v_j$ for all $1 \leq i < j \leq n$. The *restriction* of $G_\mathcal{A}$ to $V' \subseteq V$ is the CTG $G'_\mathcal{A} = \langle V', E', \lambda', queue', state' \rangle$, where $E' = E \cap (V' \times V')$, and λ', $queue'$ and $state'$ are respectively the restrictions of λ, $queue$ and $state$ to V'.

In the rest of the paper, we assume that all the CTG we consider are *well-formed*, i.e., they fulfill the following requirements:

1. For each $v \in V_T$: $state(v) \in S_{\lambda(v)}$, where $S_{\lambda(v)}$ are the states of $\mathcal{TS}_{\lambda(v)}$.

2. Each *call* vertex has at most one outgoing (queue or wait) edge, at most one incoming *wait* edge and at most one incoming *queue* edge. Each *task* vertex has at most one outgoing and at most one incoming *wait* edge.

3. For each $q \in QID$, the restriction of $G_{\mathcal{A}}$ to V_q is either empty or contains one and only one simple path of length $|V_q| - 1$. Intuitively, this ensures the well-formedness of the queues.

4. For each $q \in SQID$, there is at most one task vertex v s.t. $queue(v) = q$. This ensures that queues in $SQID$ indeed force the serial execution of their members.

For convenience, we also introduce the following notations. Let $G_{\mathcal{A}}$ be a CTG, and let q be a queue identifier of $\mathcal{A}$. Then, $head(q, G_{\mathcal{A}})$ and $tail(q, G_{\mathcal{A}})$ denote respectively the head and the tail of q in the configuration described by $G_{\mathcal{A}}$, that is, $tail(q, G_{\mathcal{A}})$ is the call vertex $v \in V_q$ that has no incoming queue edge, or $\bot$, if such a vertex does not exist; and $head(q, G_{\mathcal{A}})$ is the call vertex $v \in V_q$ that has no outgoing *queue* edge (but possibly an outgoing *wait* edge), or $\bot$, if such a vertex does not exist. Remark that, when they exist, these vertices are necessarily unique because of the well-formedness assumptions. Finally, we say that a vertex v is *unblocked* iff it has no outgoing edge, and that it is *final* iff (i) v is an *unblocked task* vertex and (ii) $state(v) = f_{\lambda(v)}$ (that is, v represents a task that has reached the final state of its transition system and is not waiting on another task).

Let us now define several operations on CTG. We will rely on these operations when defining the formal semantics of QDAS. Let $\mathcal{A}$ be a QDAS and $G_{\mathcal{A}} = \langle V, E, \lambda, queue, state \rangle$ be a CTG for $\mathcal{A}$. Then:

- for all $v \in V$: $G \setminus v$ is the restriction of G to $V \setminus \{v\}$.

- for all $\gamma \in \Gamma$ and $q \in QID$, $\mathsf{enqueue}(q, \gamma)(G_{\mathcal{A}})$ is the CTG $\langle V', E', \lambda', queue', state \rangle$ where: $V' = V \cup \{v'\}$, v' is a fresh queue vertex, $\lambda'(v') = \gamma$, $queue'(v') = q$, $state'(v') = s_{\gamma}^0$, and for all $v \in V$: $\lambda'(v) = \lambda(v)$ and $queue'(v) = queue(v)$. Finally, $E' = E \cup E_1 \cup E_2$, where: (i) $E_1 = \{(v', tail(G_{\mathcal{A}}, q))\}$ if $tail(G_{\mathcal{A}}, q) \neq \bot$ and $E_1 = \emptyset$ otherwise and (ii) if $v \in V$ is a *task* node s.t. $queue(v) = q \in SQID$, then $E_2 = \{(v', v)\}$, otherwise $E_2 = \emptyset$. Intuitively, this operation inserts a call to γ in the queue q, by creating a new vertex v' and adding an edge to maintain the FIFO ordering, if necessary (set E_1). In the case of a *serial* queue that was empty before the enqueue, a supplementary edge (in set E_2) might be necessary to ensure that v' is blocked by a currently running v which has been extracted from q.

- for all $q \in QID$, if $head(G_{\mathcal{A}}, q)$ is different from $\bot$ and equal to *unblocked*, then $\mathsf{dequeue}(q)(G_{\mathcal{A}})$ is the CTG $\langle V'_C \cup V'_T, E', \lambda, queue, state \rangle$ where $V'_C = V_C \setminus \{head(G_{\mathcal{A}}, q)\}$ and $V'_T = V'_T \cup \{head(G_{\mathcal{A}}, q)\}$. Otherwise, $head(G_{\mathcal{A}}, q) = \bot$ and $\mathsf{dequeue}(q)(G_{\mathcal{A}})$ is undefined. Intuitively, this operation removes the first (with respect to the FIFO ordering) block from q and turns the corresponding *call* vertex $head(G_{\mathcal{A}}, q)$ into a *task* vertex, meaning that the block is now running as a task.

- for all $\delta = (s, a, s') \in \Delta$, $\mathsf{step}(\delta)(G_{\mathcal{A}})$ is a *set* of CTG defined as follows. $\langle V, E, \lambda, queue, state' \rangle \in \mathsf{step}(\delta)(G_{\mathcal{A}})$ iff there exists an *unblocked* $v \in V_T$ s.t. $state(v) = s$, $state'(v) = s'$ and for all $v' \neq v$: $state'(v') = state(v')$. Remark that

$\mathsf{step}(\delta)(G_{\mathcal{A}})$ can be empty. Intuitively, each graph in $\mathsf{step}(\delta)(G_{\mathcal{A}})$ corresponds to the firing of an a-labeled transition by a task that is not blocked.

- for all *unblocked* $v \in V \cup \{\bot\}$, all $v' \in V$: $\mathsf{letwait}(v, v')(G_{\mathcal{A}})$ is either the CTG $G_{\mathcal{A}}$ if $v = \bot$, or the CTG $\langle V, E \cup (v, v'), \lambda, \textit{queue}, \textit{state} \rangle$ if $v \neq \bot$. Intuitively, this operation adds a wait edge between nodes v and v' when $v \neq \bot$, and does not modify the CTG otherwise.

Semantics of QDAS: For a QDAS $\mathcal{A}$ with set of variables $\mathcal{X}$, a *configuration* is a pair $(G, \mathbf{d})$, where G is a CTG of $\mathcal{A}$ and $\mathbf{d} \in \mathsf{vals}(\mathcal{X})$. The operational semantics of $\mathcal{A}$ is given as a transition system $[\![\mathcal{A}]\!]$ whose states are configurations of $\mathcal{A}$; and whose transitions reflect the semantics of the actions labeling the transitions of the QDAS. Formally, given a QDAS $\mathcal{A} = \langle CQID, SQID, \Gamma, main, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$, $[\![\mathcal{A}]\!]$ is the labeled transition system $\langle C, c^0, \widetilde{\Sigma}, \Longrightarrow \rangle$ where: (i) C contains all the pairs $(G, \mathbf{d})$ where $\mathbf{d} \in \mathsf{vals}(\mathcal{X})$, and G is a CTG of $\mathcal{A}$, (ii) $c^0 = (G^0, \mathbf{d}^0)$ with $\mathbf{d}^0(x) = d_0$ for all $x \in \mathcal{X}$, and $G^0 = \langle \{v^0\}, \emptyset, \lambda, \textit{queue}, \textit{state} \rangle$, where v^0 is a *task node*, $\lambda(v^0) = main$, $\textit{state}(v^0) = s_{\text{main}}^0$ and $\textit{queue}(v^0) = \bot$, (iii) $\widetilde{\Sigma} = \Sigma \cup \{\varepsilon\}$ and (iv) $\big((G, \mathbf{d}), a, (G', \mathbf{d}')\big) \in \Longrightarrow$ iff one of the following holds:

Async. dispatch: $a = \mathtt{dispatch_a}(q, \gamma)$, $\mathbf{d}' = \mathbf{d}$, and there are $\delta = (s, a, s') \in \Delta$ and $G'' \in \mathsf{step}(\delta)(G)$ s.t.: $G' = \mathsf{enqueue}(q, \gamma)(G'')$.

Sync. dispatch: $a = \mathtt{dispatch_s}(q, \gamma)$, $\mathbf{d}' = \mathbf{d}$ and there are $\delta = (s, a, s') \in \Delta$ and $G'' \in \mathsf{step}(\delta(G))$ s.t.: $G' = \mathsf{letwait}(v, v')\big(\mathsf{enqueue}(q, \gamma)(G'')\big)$ where v is the node whose *state* has changed during the step operation, and v' is the fresh node that has been created by the $\mathsf{enqueue}$ operation. That is, a queue vertex v' labeled by γ is added to q and a *wait* edge is added between the node v representing the task that performs the *synchronous* dispatch, and v', as the dispatch is *synchronous*.

Test: $a = g \in \mathsf{guards}(\mathcal{X})$, $\mathbf{d}' = \mathbf{d}$, $\mathbf{d} \models g$, and there is $\delta = (s, a, s') \in \Delta$ s.t. $G' \in \mathsf{step}(\delta)(G)$.

Assignment: $a = x \leftarrow v \in \mathsf{assign}(\mathcal{X})$, $\mathbf{d}'(x) = v$, for all $x' \neq x$: $\mathbf{d}'(x) = \mathbf{d}(x)$ and there is $\delta = (s, a, s') \in \Delta$ s.t. $G' \in \mathsf{step}(\delta)(G)$.

Scheduler action: $a = \varepsilon$, $\mathbf{d}' = \mathbf{d}$ and:

- either there is a final vertex v s.t. $G' = G \setminus v$;
- or there is $q \in CQID$ s.t. $\textit{head}(q, G) \neq \bot$ and $G' = \mathsf{dequeue}(q)(G)$. That is, the scheduler schedules a block (represented by v) from a concurrent queue.
- or there is $q \in SQID$ s.t. $\textit{head}(q, G) = v$, $G' = \mathsf{letwait}(\textit{head}(q, G''), v)(G'')$, $G'' = \mathsf{dequeue}(q)(G)$ and v is *unblocked*. That is, the scheduler schedules a block (represented by v) from the serial queue q. As the queue is serial, a *wait* edge is inserted between the next waiting block in q (now represented by $\textit{head}(q, G'')$) and v.

A *run* ρ of a QDAS is a finite alternating sequence $c_0 a_1 c_1 a_2 \ldots a_n c_n$ of configurations and actions where $(c_i, a_{i+1}, c_{i+1}) \in \Longrightarrow$ for all $0 \leq i < n$ and $c_0 = c^0$. A configuration c is *reachable* in $\mathcal{A}$ iff there exists a run $c_0 a_1 c_1 a_2 \ldots a_n c_n$ of $\mathcal{A}$ s.t. $c_n = c$. We denote by $Reach(\mathcal{A})$ the set of all reachable configurations of $\mathcal{A}$.

The decision problem on QDAS we consider in this work is the *Parikh coverability problem*. The *Parikh image* $\mathsf{Parikh}(G)$, of a CTG G of $\mathcal{A}$ with set of states S, is a function $f : S \to \mathbb{N}$, s.t. for all $s \in S$: $f(s) = |\{v \in V \,|\, state(v) = s\}|$. Given two Parikh images $\mathsf{Parikh}(G)$ and $\mathsf{Parikh}(G')$, we let $\mathsf{Parikh}(G) \preceq \mathsf{Parikh}(G')$ iff for all $s \in S$: $\mathsf{Parikh}(G)(s) \leq \mathsf{Parikh}(G')(s)$. Then, the Parikh coverability problem asks, given a QDAS $\mathcal{A}$ with set of locations S and a function $f : S \mapsto \mathbb{N}$, whether there is $c = (G, \mathbf{d}) \in Reach(\mathcal{A})$ s.t. $f \preceq \mathsf{Parikh}(G)$. When the answer to this question is 'yes', we say that f is *Parikh-coverable* in $\mathcal{A}$. As recalled in the introduction, meaningful verification questions can be reduced to this problem. For instance, consider a *mutual exclusion* question, asking whether it is possible to reach, in a QDAS $\mathcal{A}$, a configuration in which at least two tasks are executing the same block γ and are in the same control state s. This can be encoded into an instance of the Parikh coverability problem, where $f(s) = 2$ and $f(s') = 0$ for all $s' \neq s$.

4 The Parikh coverability problem

In this section, we study the Parikh coverability problem from a computational point of view. As expected, this problem is undecidable in general. However, when restricting the types of queues and dispatches that are allowed, it is possible to retain decidability. In these cases, we characterize the complexity of the problem. Formally, we consider the following subclasses of QDAS. A QDAS $\mathcal{A}$ with set of transitions Δ, set of serial queues $SQID$ and set of concurrent queues $CQID$, is *synchronous* iff there exists no $(s, a, s') \in \Delta$ with $a \in \{\texttt{dispatch\_a}\} \times QID \times \Gamma$; it is *asynchronous* iff there exists no $(s, a, s') \in \Delta$ with $a \in \{\texttt{dispatch\_s}\} \times QID \times \Gamma$; it is *concurrent* iff $SQID = \emptyset$ and $CQID \neq \emptyset$; it is *serial* iff $CQID = \emptyset$ and $SQID \neq \emptyset$; it is *queueless* iff $CQID = SQID = \emptyset$.

Queueless QDAS: In a queueless QDAS, there is no dispatch possible, so the only task that can execute at all time is the `main` one. Thus, configurations of queueless QDAS can be encoded as tuples $(s, \mathbf{d})$, where s is a state of `main`, and $\mathbf{d}$ is a valuation of the variables. Hence queueless QDAS are essentially LTS with variables over a finite data domain, thus:

Proposition 2 *The Parikh coverability is* PSPACE-C *for queueless* QDAS.

Synchronous QDAS: In synchronous QDAS, there is no concurrency, i.e. in all reachable configurations there is at most one running task that can execute a transition. All the other tasks have necessarily performed a *synchronous* dispatch and are thus blocked. More precisely, in every reachable configuration $(G, \mathbf{d})$ of a synchronous QDAS, G is of one of the forms depicted in Fig. 3 (i.e. $v_0, \ldots, v_{n-1} \in V_T$ and either $v_n \in V_T$ or $v_n \in V_C$). When the current CTG is of the form Fig. 3(a), the only possible
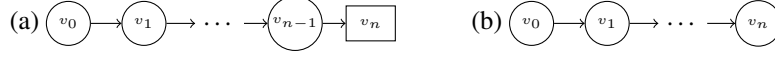
Figure 3: The two possible forms of reachable CTGs in a synchronous QDAS

action is that the scheduler starts running v_n's block and we obtain a graph of the form Fig. 3(b). In the case where the CTG is of the form (b), either v_n terminates, which removes v_n from the CTG, or v_n executes an internal action, which does not change the shape of the CTG, or v_n does a synchronous call. The latter adds a call vertex as successor of v_n which will be directly scheduled. Thus, we can assume in the following that the combined action of $\mathtt{dispatch_s}$ and scheduling the dispatched block is atomic in synchronous QDAS. Thus, intuitively, all CTGs behave similar to the content of a *pushdown stack*: a termination corresponds to a pop, a synchronous call corresponds to a push.

Let $\mathcal{A} = \langle CQID, SQID, \Gamma, main, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ be a synchronous QDAS. Let us now build a PDS with data $\mathcal{P}_\mathcal{A} = \langle Y, \mathcal{X}, y^0, S, \Sigma_\mathcal{P}, \Delta_\mathcal{P} \rangle$ to simulate $\mathcal{A}$:

- the set of states is $Y = S \cup \{\varepsilon\}$ and the initial state is $y^0 = s^0_{main}$

- $\Sigma_\mathcal{P} = (\{\mathtt{push}, \mathtt{pop}\} \times S) \cup \{\mathtt{empty?}\} \cup \mathsf{guards}(\mathcal{X}) \cup \mathsf{assign}(\mathcal{X})$

- $(y, a, y') \in \Delta_\mathcal{P}$ iff

    - either $a \in \mathsf{guards}(\mathcal{X}) \cup \mathsf{assign}(\mathcal{X})$ and $(y, a, y') \in \Delta$
    - or $a = \mathtt{push}(s')$, $(s, \mathtt{dispatch_s}(q, \gamma), s') \in \Delta$ and $y' = s^0_\gamma$
    - or $a = \mathtt{pop}(s)$, $y \in F$ and $y' = s$
    - or $a = \mathtt{empty?}$, $y = f_{main}$, and $y' = \varepsilon$.

Thus, at all times, the current location of $P_\mathcal{A}$ encodes the current location of the (single) running task in $\mathcal{A}$, and the stack content records the sequence of synchronous dispatches, as described above. A guard or assignment in $\mathcal{A}$ is kept as is in $P_\mathcal{A}$. A synchronous dispatch $(s, (\mathtt{dispatch_s}, q, \gamma), s')$ in $\mathcal{A}$ is simulated by a push of s' (to record the local state that has to be reached when the callee terminates) and moves the current state of $P_\mathcal{A}$ to the initial state of γ. The termination of a block is simulated by a pop (and we use the $\mathtt{empty?}$ action for the termination of $main$).

Formally, the relationship between $\mathcal{A}$ and $\mathcal{P}_\mathcal{A}$ is given by the $\triangleright$ operator. Let G be a CTG of one of the forms given in Fig. 3 (with vertices $v_0, v_1, \ldots v_n$), and let $w = w_0 w_1 \cdots w_n$ be a word in S^*. Then, w *encodes* G, written $w \triangleright G$, iff for all $0 \le i \le n$: $w_i = state(v_i)$ (and $\varepsilon \triangleright G$ iff G is empty). Moreover, a configuration $c = (G, \mathbf{d})$ of $\mathcal{A}$ is encoded by a configuration $x = (y, w, \mathbf{d}')$ (written $x \triangleright c$) of $\mathcal{P}_\mathcal{A}$ iff $w \cdot y \triangleright G$ and $\mathbf{d} = \mathbf{d}'$. Then:

Proposition 3 *Let $\mathcal{A}$ be a QDAS, and let $\mathcal{P}_\mathcal{A}$ be the PDS with data built from $\mathcal{A}$. There is a run $\rho = c_0 a_1 c_1 \ldots a_n c_n$ of $\mathcal{A}$ if and only if there is a run $\pi = x_0 a'_1 x_1 \ldots a'_n x_n$ in $\mathcal{P}_\mathcal{A}$ s.t. for all $0 \le i \le n$: $x_i \triangleright c_i$.*

The previous proposition allows to derive results on the reachability problem of QDAS. However, we are interested in the Parikh coverability problem. Let f be a Parikh image of $\mathcal{A}$. Then, by Proposition 4, looking for a reachable configuration of $\mathcal{A}$ that covers f amounts to finding a reachable configuration $(s_i, w_i, \mathbf{d}_i)$ of $\mathcal{P}_\mathcal{A}$ s.t. the

Parikh image P of $w_i \cdot s_i$ is s.t. $f \preceq P$ (as the CTG is encoded by $w_i \cdot s_i$). To achieve this, we augment $P_\mathcal{A}$ with a *widget* that works as follows. In any location of $P_\mathcal{A}$, we can jump non-deterministically to the widget. This jump pushes the current location to the stack so that, when entering the widget, the CTG is completely described by the stack content only. Then, the widget pops all the values from the stack, and checks that at least $f(s)$ symbols s are present on the stack. The widget jumps to an accepting state iff it is the case. We call $\mathcal{P}_{\mathcal{A},f}$ the resulting PDS. Clearly, one can build such a widget for all f, and this effectively reduces the Parikh coverability problem of QDAS to the location reachability problem of PDS. Moreover, for all f, the widget is of size exponential in $|S|$ and exponential in the binary encoding of $max_{s \in S} f(s)$. Hence, building $\mathcal{P}_{\mathcal{A},f}$ requires exponential time:

Proposition 4 *Given a synchronous* QDAS *$\mathcal{A}$ with states S and a function $f : S \to \mathbb{N}$, one can generate a* PDS *$\mathcal{P}_{\mathcal{A},f}$ of size exponential in $\mathcal{A}$ and a state s of $\mathcal{P}_{\mathcal{A},f}$, s.t. $\mathcal{P}_{\mathcal{A},f}$ reaches s iff f is Parikh coverable in $\mathcal{A}$.*

As testing emptiness of a pushdown system is in PTIME, the Parikh coverability problem is in EXPTIME for *synchronous* QDAS (with both types of queues). A matching lower bound is obtained by reducing the reachability question of PDS with data (see Proposition 1). This reduction requires only one *concurrent* queue, so the Parikh reachability problem is EXPTIME-hard for *synchronous concurrent* QDAS. Hence:

Theorem 1 *The Parikh coverability problem is* EXPTIME-C *for synchronous and for synchronous concurrent* QDAS.

Let us take a closer look on the dispatches that happen in runs of synchronous QDAS that have only *serial* queues. Assume a run of such a QDAS, and suppose the first dispatch performed along this run (by `main`) is $\mathtt{dispatch_s}(q,\gamma)$. As the dispatch is synchronous, `main` is blocked, and the scheduler has to dequeue γ to let the system progress. Clearly, if γ performs a synchronous dispatch $\mathtt{dispatch_s}(q,\gamma')$ to the same queue q, we reach a deadlock. Indeed, the task running γ is blocked by the synchronous dispatch of γ', but we need to wait for the termination of γ to be able to dequeue γ' from q (because q is serial). So, γ has to dispatch its blocks to other queues. For the same reason, we also reach a deadlock if a block called by γ performs a synchronous dispatch into q. We conclude that, in all reachable CTG, the following holds for all queues: either the queue contains one block and there is no running task from this queue, or the queue is empty, and there is at most one running task from this queue. Hence, all the reachable CTG have at most $|SQID| + 2$ vertices. Thus, the pushdown systems used in all previous constructions have bounded stack height and we can apply the emptiness test on a finite state system when proving Proposition 4. The lower bound can be derived from Proposition 2.

Theorem 2 *The Parikh coverability problem is* PSPACE-C *for serial synchronous* QDAS.

Concurrent asynchronous QDAS: Let us now establish a relationship between *concurrent asynchronous* QDAS and Petri nets tho prove that the Parikh coverability problem is EXPSPACE-complete. We first show how to reduce the QDAS Parikh coverability problem to the Petri net coverability problem. Let $\mathcal{A} = \langle CQID, \emptyset, \Gamma, \mathtt{main}, \mathcal{X}, \Sigma,$

$(\mathcal{TS}_\gamma)_{\gamma \in \Gamma}\rangle$ be a concurrent asynchronous QDAS on domain $\mathbb{D}$. We associate to $\mathcal{A}$ the PN $N_\mathcal{A} = \langle P, T, m_0 \rangle$ where: $P = (\mathcal{X} \times \mathbb{D}) \cup S$, $t = (I_t, O_t) \in T$ iff one of the following holds:

Async. dispatch: there exists $(s, (\mathtt{dispatch_a}, q, \gamma), s') \in \Delta$ s.t. $I_t(p) = 1$ iff $p = s$ and $O_t(p) = 1$ iff $p \in \{s', s^0_\gamma\}$.

Test: there exists $(s, x = v, s') \in \Delta$ where $x = v \in \mathsf{guards}\,(\mathcal{X})$ s.t. $I_t(p) = 1$ iff $p \in \{(x, v), s\}$ and $O_t(p) = 1$ iff $p \in \{(x, v), s'\}$.

Assignment: there exists $(s, a, s') \in \Delta$ with $a = x \leftarrow v \in \mathsf{assign}\,(\mathcal{X})$ and $v' \in \mathbb{D}$ s.t. $I_t(p) = 1$ iff $p \in \{(x, v'), s\}$, $O_t(p) = 1$ iff $p \in \{(x, v), s'\}$.

Task termination: there exists $\gamma \in \Gamma$ s.t. $I_t(p) = 1$ iff $p = f_\gamma$ and for all $p \in P$: $O_t(p) = 0$.

and finally, $m_0(s^0_{main}) = 1$, for all $x \in \mathcal{X}$: $m_0(x, d_0) = 1$ and for all $p \in P \setminus \{s^0_{main}\} \cup \{(x, d_0) \mid x \in \mathcal{X}\}$: $m_0(p) = 0$.

The intuition behind the construction is as follows. Each place $s \in S$ counts how many blocks are currently running and in state s. Each place (x, d) encodes the fact that variable x contains value d in the current valuation. Remark that we have no place to encode the contents of the queue, as the dispatch of block γ directly creates a new token in s^0_γ. This encoding is, however, correct with respect to the *Parikh coverability problem*, as $\mathsf{Parikh}(G)$ does not distinguish between a block γ that is waiting in a queue, and a task executing γ in its initial state. Thus:

Proposition 5 *For all concurrent asynchronous QDAS $\mathcal{A}$ with set of location S, we can build, in polynomial time, a PN $N_\mathcal{A}$ s.t. f is Parikh-coverable in $\mathcal{A}$ iff $m \in Cover(N_\mathcal{A})$, where m is the marking s.t. for all $s \in S$: $m(s) = f(s)$ and for all $p \in P \setminus S$: $m(p) = 0$.*

Let us now reduce the Petri net coverability problem to the QDAS Parikh coverability problem. Let $N = \langle P, T, m_0 \rangle$ be a PN. We associate to N the concurrent asynchronous QDAS $\mathcal{A}_N = \langle CQID, \emptyset, \Gamma, \mathtt{main}, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$, on the finite domain $\mathbb{D} = \{0, 1\}$, where $CQID = \{C\}$, $\Gamma = \{main, trans\} \cup P$, $\mathcal{X} = \{v_p \mid p \in P\}$ and $(\mathcal{TS}_\gamma)_{\gamma \in \Gamma}$ is given by the pseudo-code in Fig. 4 (this construction is an extension of a construction found in [9]). We assume that, for $\gamma \in \{\mathtt{trans}, \mathtt{main}\}$: s^ℓ_γ is the location of $\mathcal{TS}_\gamma$ that is reached when the control reaches line ℓ. Let $G = \langle V, E, \lambda, queue, state \rangle$ be a CTG for $\mathcal{A}_N$, and let m be a marking of N. Then, we say that G *encodes* m, written $G \rhd m$ iff (i) $\mathsf{Parikh}(G)(s^{14}_{\mathtt{trans}}) = \mathsf{Parikh}(G)(s^8_{\mathtt{main}}) = 1$, (ii) for all $p \in P$: $\mathsf{Parikh}(G)(s^0_p) = m(p)$ and (iii) for all $p \in P$, for all $s \in S_p \setminus \{s^0_p\}$: $\mathsf{Parikh}(G)(s) = 0$. Thus, intuitively, a CTG G encodes a marking m iff $\mathtt{main}$ is at line 8, $\mathtt{trans}$ is at line 14, $m(p)$ counts the number of $\mathtt{p}$ blocks that are either in C or executing but at their initial state, and there are no $\mathtt{p}$ blocks that are in state s^{mid}_p or s^{fin}_p.

The intuition behind the construction is as follows. Each run of the QDAS $\mathcal{A}_N$ starts with an initialization phase, where $\mathtt{main}$ initializes all the v_p variables to 0 and dispatches, for all $p \in P$, k_p blocks $\mathtt{p}$ with $k_p \leq m_0(p)$, then dispatches a call to

```
 1  def main():
 2    for each p ∈ P:
 3      v_p := 0
 4      select k_p ∈ {0, ..., m_0(p)}
 5      for i = 0...k_p:
 6        dispatch_a(C, p())
 7      dispatch_a(C, trans())
 8    while(true): do nothing

10  block p(): // For all p ∈ P
11    while(v_p = 0): do nothing
12    v_p := 0

13  block trans():
14    while(true):
15      select t = (I_t, O_t) ∈ T
16      for each p ∈ P s.t. I_t(p) = 1:
17        v_p := true
18      while(∃p ∈ P: v_p = 1): do nothing
19      for each p ∈ P s.t. O_t(p) = 1:
20        dispatch_a(C, p())
```

Figure 4: Encoding of PN coverability $\langle P, T, m_0 \rangle$ by a QDAS

trans. At that point, the only possible action is that the scheduler dequeues all the blocks. All the p tasks are then blocked, as they need that $v_p = 1$ to proceed and terminate. Then, trans cyclically picks a transition t, sets to 1 all the variables v_p s.t. t consumes a token in p, and waits that all the v_p variables return to 0. This can only happen because *at least* $I_t(p)$ p tasks have terminated, for all $p \in P$. So, when trans reaches line 19, the encoded marking has been decreased by *at least* I_t. Remark that more than $I_t(p)$ p tasks could terminate, as they run concurrently, and the lines 11 and 12 do not execute atomically. Then, trans dispatches one new p block iff t produces a token in p. This increases the encoded marking by O_t, so the effect of one iteration of the main while loop of trans is to simulate the effect of t, plus a possible token loss. Hence, the resulting marking is guaranteed to be in *Cover(N)* (but maybe not in *Reach(N)*). This is formalized by the following proposition:

Proposition 6 *For all* PN *N, we can build, in polynomial time, a concurrent asynchronous* QDAS *$\mathcal{A}_N$ s.t. $m \in Cover(N)$ iff there exists $(G, \mathbf{d}) \in Reach(\mathcal{A}_N)$ with $G \rhd m$.*

Theorem 3 *The Parikh coverability problem is* EXPSPACE-*complete for concurrent asynchronous* QDAS.

Asynchronous Serial QDAS: Let us show that for the class of QDAS with one serial queue, and where asynchronous dispatches are allowed, the Parikh coverability problem is *undecidable*. We establish this by a reduction from the control-state reachability problem in a fifo system. Let $F = \langle S_F, s_F^0, M, \Delta_F \rangle$ be a fifo system and let $c \in S_F$ be a control state whose reachability has to be tested. We build the asynchronous serial QDAS $\mathcal{A}_F = \langle \emptyset, \{q\}, \Gamma, \mathtt{main}, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ on domain $\mathbb{D} = M \cup S_F \cup \{\varepsilon\}$, where $\Gamma = M \cup \{\varepsilon, \mathtt{main}\}$, $\mathcal{X} = \{\mathtt{state}, \mathtt{head}\}$ and the $\mathcal{TS}_\gamma$ are sketched in Fig. 5.

Intuitively, runs of $\mathcal{A}_F$ simulate the runs of F, by encoding the current state of F in variable state and the content of F's queue into the content of the serial queue q. More precisely, it easy to check that, once main has reached line 8, all the CTG that are reached in $\mathcal{A}_F$ are of either shapes depicted in Fig. 5, for $\{m_1, \ldots, m_n, m\} \subseteq M \cup \{\varepsilon\}$. That is, there are at most two running tasks: main and possibly one task running a m block (for m $\in M \cup \{\varepsilon\}$), that has to terminate to allow a further dequeue from q. This is because q is a serial queue and all the dispatches are asynchronous. When the CTG is of shape (b), the duty of the running m block is to simulate a step of F. It runs an infinite while loop (line 11 onwards – ignore the test at line 10 for the moment), that

14

```
1  global state, head                    9  def m(): //for all m ∈ M ∪ {ε}
2  global s_queue q                      10    if (head≠ m): goto 20
                                         11    while(true):
4  def main():                          12      if (state = c): goto 21
5    state := s_F^0                      13      select (s, a, s') ∈ Δ_y
6    head := ε                          14      if (s ≠state): goto 20
7    dispatch_a(q, ε)                    15      state := s'
8    while(true): do nothing            16      if (a =!n): dispatch_a(q, n)
                                         17      else if (a =?n):
                                         18        head := n
                                         19        terminate
┌────────────────────────────────────┐ 20    while(true): do nothing // wrong guess
│ Note that the reachability of a state c of the │ 21    while(true): do nothing // c is reached
│ fifo system is explicitly coded into the control │
│ structure.                          │
└────────────────────────────────────┘
```

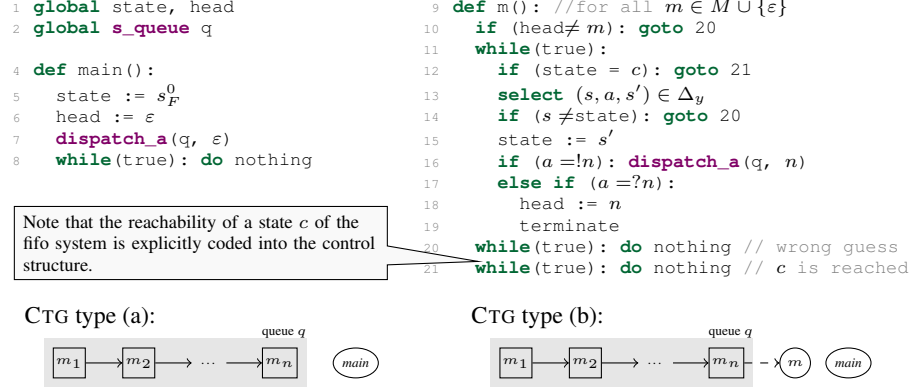CTG type (a):

CTG type (b):

Figure 5: Fifo system encoding into a serial asynchronous QDAS/ two types of CTG in this case

(i) tests whether c has been reached (line 12) and jumps to line 21 if it is the case; (ii) guesses a transition (s, a, s') of F; and (iii) checks that the guessed transition is indeed fireable from the current configuration of F, and, if yes, simulate it. This consists in, first testing that s is the current state (line 14). If not, the block jumps to the infinite loop of line 20, which ends the simulation. Otherwise, the current state is updates to s', and the channel operation is then simulated. A send of message n is simulated (line 16) by an asynchronous dispatch of block n to q. The simulation of a receive of n from q is more involved, as only the scheduler can decide to dequeue a block from q, and this can happen only if the current running block terminates (line 19). Still, we have to check that message n is indeed in the head of q. This is achieved by setting global variable head to n, and letting the next dequeued block check that itself encodes the value stored into head. This is performed at line 10. If this test is not satisfied, the block jumps to the infinite loop of line 20, and the simulation ends. Otherwise, it proceeds with the simulation. Thus, in all reachable configurations of $\mathcal{A}_F$, a block m (with $m \in M \cup \{\varepsilon\}$ will reach line 21 iff c is reachable in F. This effectively reduces the control location reachability of fifo systems to the Parikh coverability problem of serial asynchronous QDAS.

Theorem 4 *The Parikh coverability problem is undecidable for asynchronous* QDAS *with at least one serial queue.*

Concurrent QDAS: Let us show that, once we allow both synchronous and asynchronous dispatches in a *concurrent* QDAS, the Parikh coverability problem becomes undecidable too. For that purpose, we reduce the reachability problem of two counter systems to the Parikh coverability problem of concurrent QDAS. Let us give the intuition of the construction. For each two counter system $\mathcal{P}$, we construct a QDAS $\mathcal{A}_\mathcal{P}$ s.t. all reachable CTG in $\mathcal{A}_\mathcal{P}$ encode configurations of $\mathcal{P}$ and are of the form depicted in Fig. 6. That is, (after an initialization phase), there are always three tasks that are unblocked: a *main* task to simulate $\mathcal{P}$'s control structure, and, for each $i = \{1, 2\}$,
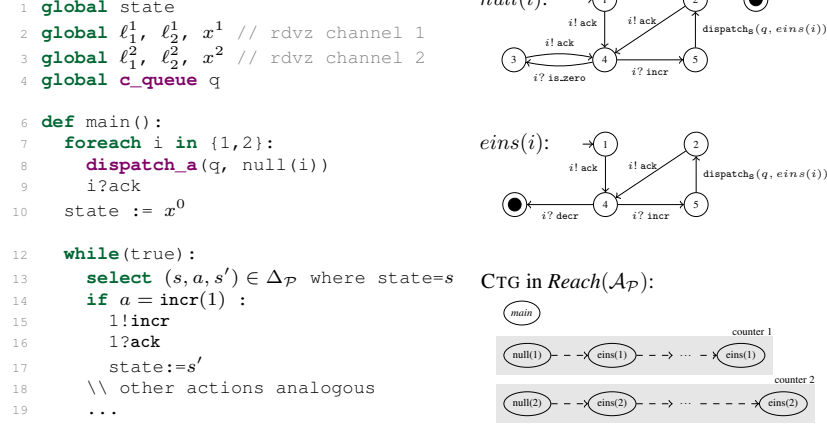
```
1  global state
2  global ℓ₁¹, ℓ₂¹, x¹ // rdvz channel 1
3  global ℓ₁², ℓ₂², x² // rdvz channel 2
4  global c_queue q

6  def main():
7    foreach i in {1,2}:
8      dispatch_a(q, null(i))
9      i?ack
10   state := x⁰

12   while(true):
13     select (s,a,s') ∈ Δ_P where state=s
14     if a = incr(1) :
15       1!incr
16       1?ack
17       state:=s'
18     \\ other actions analogous
19     ...
```

$null(i)$:

$eins(i)$:

CTG in $Reach(\mathcal{A}_P)$:

Figure 6: From a two counter system to a QDAS: *main* and $null(i)$, $zero(i)$ for $i = 1, 2$

either a task $eins(i)$ or a task $null(i)$. If the task $null(i)$ is unblocked, then counter i is zero in the current configuration of $\mathcal{P}$. Otherwise, the current valuation of counter i is encoded by the number of $eins(i)$ tasks in the CTG. Remark that, as in the case of synchronous QDAS, the parts of the CTG that encode each counter behave as pushdown stacks. Finally, the control location of $\mathcal{P}$ is recorded in global variable state.

The actual operations on the counters will be simulated by the $eins(i)$ and $null(i)$ running tasks. As *main* simulates the control structure, we need to synchronize *main* with those $eins(i)$ and $null(i)$ tasks. Let us explain intuitively how we can achieve *rendezvous* synchronization between running tasks using global variables of QDAS. Consider a QDAS with three global variables ℓ_1, ℓ_2 ranging over the Boolean and a global variable x ranging over a finite set of 'messages' M. Let γ_1 and γ_2 be two blocks whose LTS are:

γ_1: $s_0 \xrightarrow{\ell_1 = 1} \bigcirc \xrightarrow{x \leftarrow m} \bigcirc \xrightarrow{\ell_2 \leftarrow 1} \bigcirc \xrightarrow{\ell_1 = 0} \bigcirc \xrightarrow{\ell_2 \leftarrow 0} s_5$ (for $m \in M$)

γ_2: $s_0' \xrightarrow{\ell_1 \leftarrow 1} \bigcirc \xrightarrow{\ell_2 = 1} \bigcirc \xrightarrow{x = m} \bigcirc \xrightarrow{\ell_1 \leftarrow 0} \bigcirc \xrightarrow{\ell_2 = 0} s_5'$

Assume a configuration c of the QDAS where $\ell_1 = \ell_2 = 0$ and where two distinct tasks are running γ_1 and γ_2, are unblocked, and are in s_0 and s_0' respectively. Assume that no other task can access ℓ_1, ℓ_2 and x. It is easy to check that, from c, there is only one possible interleaving of the transitions of γ_1 and γ_2. So if γ_2 reaches s_5' from c, then γ_1 must have reached s_5, and the $x = m$ test in γ_1 has been fired *after* the $x \leftarrow m$ assignment in γ_2. This achieves rendezvous synchronisation between γ_1 and γ_2, with the passing of message m. This can easily be extended to rendezvous via different "channels", by duplicating those global variables. So, we extend the syntax of QDAS by allowing transitions of the form $(s_0, c!m, s_5)$ and $(s_0', c?m, s_5')$ (for $m \in M$) to denote respectively a send and a receive of message m on a rendezvous channel c.

We rely on this mechanism to let *main* send operations to be performed on the counters to the $null(i)$ and $eins(i)$ running tasks. More precisely, for a 2Cs $\mathcal{P} = \langle X, x^0, \Sigma_\mathcal{P}, \Delta_\mathcal{P} \rangle$, we build the QDAS $\mathcal{A}_\mathcal{P} = \langle CQID, \emptyset, \Gamma, \texttt{main}, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ where $CQID = \{q\}$, $\Gamma = (\{null, eins\} \times \{1, 2\}) \cup \{main\}$, $\mathcal{X} = \{\ell_1^1, \ell_2^1, x^1, \ell_1^2, \ell_2^2, x^2\}$

where x^1, x^2 range over the domain $\{\texttt{incr}, \texttt{decr}, \texttt{is\_zero}, \texttt{ack}\}$, and the transition systems are given in Fig. 6. The variables $\mathcal{X}$ encode two channels that we call 1 and 2 in the pseudo code of Fig. 6. The *main* task runs an infinite `while` loop (line 12 onwards) that consists in guessing a transition (s, a, s') of F and synchronising, via *rendezvous* on the channels 1 and 2, with the relevant $null$ or $eins$ unblocked task, to let it execute the operation on the counter. When a $null(i)$ or $eins(i)$ receives an `incr` message, it performs a synchronous dispatch of $eins(i)$ into q to increment counter i, and acknowledges the operation to *main*, thanks to message `ack`. When an $eins$ block receives a `decr` message, it terminates, which decrements the counter. $null$ blocks cannot receive `decr` messages, so, if *main* requests a `decr` operation when the counter is zero, *main* gets blocked. This means that the guessed transition was not fireable in the currently simulated 2Cs configuration, and ends the simulation. Finally, only $null$ blocks can receive and acknowledge `is_zero` messages, so, again, *main* is blocked after sending `is_zero` to a non-zero counter. Note that we need both *asynchronous* calls to start two counters in parallel, and *synchronous* calls to encode the counter values.

It is easy to check that $\mathcal{P}$ reaches a state $x \in X$ iff f is coverable in $\mathcal{A}_\mathcal{P}$ where $f(x) = 1$ and $f(x') = 0$ for all $x \in X \setminus \{x\}$. Hence:

Theorem 5 *The Parikh coverabililty problem is undecidable for concurrent* QDAS *that use both synchronous and asynchronous dispatches.*

References

[1] Grand Central Dispatch (GCD) Reference. Technical report, Apple Inc., 2010.

[2] Concurrency Programming Guide. Technical report, Apple Inc., 2011.

[3] A. Bouajjani and M. Emmi. Analysis of recursively parallel programs. In *Proc. of POPL'12*, 2012. (to be published).

[4] A.Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. of CONCUR'97*, LNCS 1243, p.135–150. 1997.

[5] D. Brand, P. Zafiropulo. On Communicating Finite-State Machines J. ACM 30(2), 1983.

[6] R. Chadha and M. Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. In *Proc. of CONCUR'07*, LNCS 4703, p.136–150, Springer, 2007.

[7] R. Chadha and M. Viswanthan. Deciding branching time properties for asynchronous programs. *TCS*, 410(42):4169–4179, 2009.

[8] J. Esparza. Decidability and complexity of petri net problems — an introduction. In *Lectures on Petri nets I: Basic Models, Advances in Petri nets*, LNCS 1491. Springer, 1998.

[9] P. Ganty and R. Majumdar. Algorithmic verification of asynchronous programs. *CoRR*, abs/1011.0551, 2010.

[10] P. Ganty, R. Majumdar, and A. Rybalchenko. Verifying liveness for asynchronous programs. In *Proc. of POPL'09*, p.102–113. ACM, 2009.

[11] Steven M. German and A. Prasad Sistla. Reasoning about Systems with Many Processes In J. ACM-39(3)675-735, 1992.

[12] R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *Proc. of POPL'07*, p.339–350. ACM, 2007.

[13] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[14] K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *Proc. of CAV'06*, LNCS 4144, p.300–314, Springer, 2006.

A Proof for Section 2

Proposition 1 *The reachability problem is* EXPTIME-C *for* PDS *with data.*

Proof. For the upper bound, we generate a reachability-equivalent PDS (without data) by encoding all possible data valuations into the pushdown system's states. This leads to an exponential blowup of the state space. The lower bound can be derived from the reduction of the emptiness test of the intersection of a context-free language with n regular languages that is known to be EXPTIME-hard (hardness follows easily by a reduction from linearly bounded alternating Turing machines; a closely related problem, the reachability of pushdown systems with checkpoints, is shown to be EXPTIME-hard in (*).

(*) Javier Esparza, Antonín Kučera, and Stefan Schwoon: *Model checking LTL with regular valuations for pushdown systems*, in Information and Computation, 186(2):355–376, 2003.

B   Proofs of Section 4

B.1   Synchronous QDAS

Proposition 3 *Let $\mathcal{A}$ be a* QDAS, *and let $\mathcal{P}_{\mathcal{A}}$ be the* PDS *with data built from $\mathcal{A}$. There is a run $\rho = c_0 a_1 c_1 \ldots a_n c_n$ of $\mathcal{A}$ if and only if there is a run $\pi = x_0 a_1' x_1 \ldots a_n' x_n$ in $\mathcal{P}_{\mathcal{A}}$ s.t. for all $0 \le i \le n$: $x_i \rhd c_i$.*

Proof. We assume that $\mathcal{A} = \langle CQID, SQID, \Gamma, main, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ (where, as usual, S denotes the set of all states of $\mathcal{A}$), and that $\mathcal{P}_{\mathcal{A}} = \langle Y, \mathcal{X}, y^0, S, \Sigma_{\mathcal{P}}, \Delta_{\mathcal{P}} \rangle$. For convenience, we rewrite configurations $(y, w, d) \in Y \times \Sigma^* \times \mathbb{D}^{\mathcal{X}}$ of $\mathcal{P}_{\mathcal{A}}$ as pairs $(x, d) \in S^* \times \mathbb{D}^{\mathcal{X}}$ where $x = w \cdot y$ encodes both the current location and the stack content. Thus, with this new notation, we have $(x, \widehat{\mathbf{d}}) \rhd (G, \mathbf{d})$ iff $x \rhd G$ and $\mathbf{d} = \widehat{\mathbf{d}}$.

Let us prove the first implication. Let $(G_0, \mathbf{d}_0) a_1 (G_1, \mathbf{d}_1) a_2 \ldots a_n (G_n, \mathbf{d}_n)$ be a run of $\mathcal{A}$. Let us show how to build a run $(x_0, \widehat{\mathbf{d}}_0) a_1 (x_1, \widehat{\mathbf{d}}_1) a_2 \ldots a_n (x_n, \widehat{\mathbf{d}}_n)$ of $\mathcal{P}_{\mathcal{A}}$ such that $(x_i, \widehat{\mathbf{d}}_i) \rhd (G_i, \mathbf{d}_i)$ for all i.

The construction is by induction on the length of the run of $\mathcal{A}$. By construction of $\mathcal{P}_{\mathcal{A}}$, $G_0 \rhd x_0$ and $\mathbf{d}_0 = \widehat{\mathbf{d}}_0$. We now assume that there exists a prefix of the QDAS's run of length $0 \le j \le n$ of the form $(G_0, \mathbf{d}_0) \ldots (G_j, \mathbf{d}_j)$ such that there exists a run of the pushdown system $(x_0, \widehat{\mathbf{d}}_0) \ldots (x_j, \widehat{\mathbf{d}}_j)$ that fulfills the induction hypothesis. We now consider the outcome of a QDAS transition labeled a_{j+1}. We know that G_j must be a path of vertices $v_0 \ldots v_n$ connected by wait edges.

Sync. dispatch: dispatching a block γ on queue q leads to $(G_{j+1}, \mathbf{d}_{j+1})$ with $\mathbf{d}_j = \mathbf{d}_{j+1}$ and G_{j+1} is a path graph $v_0 v_1 \ldots v_n v_{n+1}$ with new distinct vertex v_{n+1} where $state(v_{n+1}) = v_\gamma^0$. We mapped the dispatch rule to a push of the current state to the pushdown and jumping to the new initial state, i.e., we go from $(x_j, \widehat{\mathbf{d}}_j)$ to $(x_{j+1}, \widehat{\mathbf{d}}_{j+1})$ where $\widehat{\mathbf{d}}_j = \widehat{\mathbf{d}}_{j+1}$ and $x_{j+1} = x_j \cdot s_\gamma^0$. Obviously, $x_{j+1} \rhd G_{j+1}$.

Test/Assignment: G_{j+1} equals G_j except for $state_j(v_n) = s$ and $state_{j+1}(v_n) = s'$ and a possible change of $\mathbf{d}_{j+1}$ according to the underlying data action. Executing the same action on $\mathcal{P}_A$ assures that $\widehat{\mathbf{d}}_{j+1} = \mathbf{d}_{j+1}$ and changing the control state of the pushdown only changes $x_j = w \cdot s$ to $x_{j+1} = w \cdot s'$; thus, $x_{j+1} \triangleright G_{j+1}$.

Termination: To apply the action G_j consists of a (non-empty) path ending in v with $state_j(v) \in F$ and $G_{j+1} = G_j \setminus v$, and $\mathbf{d}_j = \mathbf{d}_{j+1}$. Note that G_{j+1} could be possibly empty. Given a $(x_j, \widehat{\mathbf{d}}_j)$ according to the induction hypothesis, then we have to consider two cases: either $x_j = w_j \cdot y_j$ with $w_j \in S^+$ and $y_j \in S$ (i.e., there is at least one element on the stack), or $x_j = y_j \in S$ (i.e., stack is empty). In the second case, we know that $x_j \in S_{main}$ and by the induction hypothesis, that $x_j = s_{main}^0$ and G_j a path of length 1. Now, $\mathcal{P}_A$ takes the empty? transition leading to the (bottom) state ε, i.e., $x_{j+1} = \varepsilon$, hence G_{j+1} is empty and $\varepsilon \triangleright G_{j+1}$. If the stack is not empty, then we can take a poptransition such that $x_{j+1} = w \in S^+$ for $x_j = w \cdot s$, hence $x_{j+1} \triangleright G_{j+1}$. Obviously $\mathbf{d}_{j+1} = \mathbf{d}_j = \widehat{\mathbf{d}}_j = \widehat{\mathbf{d}}_{j+1}$.

(Recall that we asserted dispatch and scheduling to be atomic, so we do not need to consider other actions of the scheduler.)

The reverse direction follows analogously. □

Lemma 1 *Given a finite set S and a function $f : S \to \mathbb{N}$, then there exists a finite automaton $\mathcal{F}_f$ with alphabet S of size exponential in $|S|$ and polynomial in $max_{s \in S} f(s)$ such that $\mathcal{L}(\mathcal{F}_f) = \{w \in S^* : |w|_s \geq f(s) \text{ for all } s \in S\}$.*

Proof. Given a set S and a function $f : S \mapsto \mathbb{N}$, let $k = max_{s \in S} f(s)$ (which must exists as S is finite). Then $\mathcal{F}_f$ is the finite automaton $\langle Q, S, q^0, \Delta, q^f \rangle$ with states $Q = \{0 \ldots k\}^S$ (interpreted as an S-indexed vector of values in $0 \ldots k$), an action alphabet S, the initial state is q^0 where $q^0(s) = f(s)$, the finial state is q^f where $q^f(s) = 0$. The transitions of $\mathcal{F}_f$ are defined as follows: $(q, s, q') \in \Delta$ iff $q'(s) = q(s) - 1$ for $q(s) > 1$, else $q'(s) = q(s)$, and for all $t \in S \setminus \{s\}$ we have $q'(t) = q(t)$. Thus each transition labeled by an action s reduces the "counter" $q(s)$ by one until zero and once arrived at zero, the counter $q(s)$ remains zero for any further s action. Further, the control structure of $\mathcal{F}_f$ is acyclic (except for the loops at q^f), thus each run can visit each state in $Q \setminus \{q^f\}$.

If $w = a_1 \ldots a_n \in \mathcal{L}(\mathcal{A})$ then it was accepted by a run $q_0 a_1 q_1 \ldots a_n q_n$ where $q_0 = q^0$ and $q_n = q^f$. Due to our construction of Δ, it holds for $w = a_1 \ldots a_n$ that $|w|_s \geq q_0(s) = f(s)$ for all $s \in S$. If $w \notin \mathcal{L}(\mathcal{A})$ then there exists a run $q_0 a_1 q_1 \ldots a_n q_n$ where $q_0 = q^0$ and for $q_n \neq q^f$ it holds that there exists at least one $s \in S$ such that $q_n(s) > 0$, each transition (q_{i-1}, a_i, q_i) assures that $q_{i-1}(s) \geq q_i(s)$, hence $|w|_s < f(s)$ for at least one $s \in S$. □

Proposition 4 *Given a synchronous QDAS $\mathcal{A}$ with states S and a function $f : S \to \mathbb{N}$, one can generate a PDS $\mathcal{P}_{A,f}$ of size exponential in $\mathcal{A}$ and a state s of $\mathcal{P}_{A,f}$, s.t. $\mathcal{P}_{A,f}$ reaches s iff f is Parikh coverable in $\mathcal{A}$.*

Proof. First, we construct the PDS with data $\mathcal{P}_\mathcal{A}$ and states S as explained in the preamble of Proposition 3. Then, we translate the PDS with data to a bisimilar PDS without data $\widehat{\mathcal{P}_\mathcal{A}} = \langle \widehat{Y}, \widehat{y^0}, \widehat{\Phi}, \widehat{\Sigma}, \widehat{\Delta} \rangle$ by encoding all possible valuations of variables into the PDS's states by a standard product construction, i.e., $\widehat{Y} = S \times (\mathcal{X} \times \mathbb{D})$. Given $y \in \widehat{Y}$, let $S(y) \in S$ denote the original state component. Note: $\widehat{\mathcal{P}_\mathcal{A}}$ is at most exponentially larger as $\mathcal{P}_\mathcal{A}$ and this construction does not change the pushdown system's behaviour with respect to the stack but only internal actions.

Second, from the function f, we construct the automaton $\mathcal{F}_f = \langle Q, S, q^0, \Delta_\mathcal{F}, q^f \rangle$ as explained in Lemma 1 above.

Finally, we define the PDS $\mathcal{P}_{\mathcal{A},f} = \langle Y, y^0, \Phi, \Sigma, \Delta_{\mathcal{A},f} \rangle$ as follows

- states are $Y = \widehat{Y} \cup Q$ (assuring disjointness by relabeling when necessary)
- $y^0 = \widehat{y^0}$ is the initial state
- $\Phi = \widehat{\Phi}$ is the stack alphabet (where $\widehat{\Phi} = S$ due to the above construction)
- $\Sigma = \widehat{\Sigma} \cup \{\varepsilon\}$
- a tuple (y, a, y') is a rule in $\Delta_{\mathcal{A},f} \subseteq Y \times \Sigma \times Y$ iff one of the following holds

  - $(y, a, y') \in \widehat{\Delta}$ (include all transition rules of $\widehat{\mathcal{P}_\mathcal{A}}$);
  - $a = \mathrm{pop}(s)$ for $s \in \Phi$ and $(q, s, q') \in \Delta_\mathcal{F}$ (include rules of $\mathcal{F}_f$ and change an s action to $\mathrm{pop}(s)$ for $s \in S$);
  - $y \in \widehat{Y}$, $a = \mathrm{push}(z)$ for $z = S(y)$, and $y' = q^0$ (connect all states in $\widehat{Y}$ with the initial state of $\mathcal{F}_f$, additionally stocking the current "state"-component on the stack).

Note that $\mathcal{P}_{\mathcal{A},f}$ is of size exponential with respect to both the QDAS and f due to serial composition.

We now have to show that if there is a run in $\mathcal{P}_{\mathcal{A},f}$ that reaches the state q^f, then there exists configuration $c = (G, \mathbf{d})$ of $\mathcal{A}$ such that $f \preceq \mathsf{Parikh}(G)$.

Assume that there exists a run of $\mathcal{P}_{\mathcal{A},f}$ reaching q^f, then it must be of the following form $\langle x_0, a_1, x_1, \ldots, a_k, x_k, a_{k+1}, x_{k+1}, a_{k+2}, \ldots, a_n, x_n \rangle$ where $x_i = (y_i, w_i) \subseteq Y \times S^*$ are the corresponding infinite transition system configurations. Further, $y_0 = y^0$, $y_n = q^f$, $y_{k+1} = q^0$, and $\langle y_1 \ldots y_k \rangle$ is a subrun that only uses states in $\widehat{Y}$ as well as transitions in $\widehat{\mathcal{P}_\mathcal{A}}$; $\{y_{k+1}, \ldots, y_n\} \subseteq Q$ and the corresponding transitions are derived from $\Delta_\mathcal{F}$, as well as $a_{k+1} = \mathrm{push}(S(y_k))$.

Let us take a closer look on the first part of the run: $\langle y_0, a_1, \ldots, a_n, x_k \rangle$ is equivalent to a run of $\widehat{\mathcal{P}_\mathcal{A}}$ that reaches a configuration x_k. The latter is, following Proposition 3, similar to a run of the original QDAS $\mathcal{A}$ that reaches a configuration $c = (G, \mathbf{d})$ where $G \triangleright y_k \cdot S(y_k)$. Thus, $c \in Reach(\mathcal{A})$.

The transition $(x_k, \mathrm{push}(S(y_k)), x_{k+1})$ now transfers the encoding of G to the stack, i.e., $w_{k+1} = y_k \cdot S(y_k)$. All other information on data encoded in y_k is lost in this step.

Now, by Lemma 1 we know that the subrun $\langle x_{k+1}, a_{k+2}, \ldots, a_n, x_n \rangle$ leading to the final state of $\mathcal{F}_f$ assures that $|w_{k+1}|_s \geq f(s)$ for all $s \in S$. Hence, for the previously found $c = (G, \mathbf{d}) \in Reach(\mathcal{A})$ it holds that $f \preceq \mathsf{Parikh}(G)$. □

B.2 Asynchronous Concurrent QDAS

Proposition 5 *For all concurrent asynchronous* QDAS $\mathcal{A}$ *with set of location S, we can build, in polynomial time, a* PN $N_{\mathcal{A}}$ *s.t. f is Parikh-coverable in $\mathcal{A}$ iff $m \in Cover(N_{\mathcal{A}})$, where m is the marking s.t. for all $s \in S$: $m(s) = f(s)$ and for all $p \in P \setminus S$: $m(p) = 0$.*

The proof of the proposition relies on the following lemma, showing that $N_{\mathcal{A}}$ can simulate precisely the sequence of Parikh images that are reachable in $\mathcal{A}$. Let $(G, \mathbf{d})$ be a configuration of $\mathcal{A}$, and let m be marking of $N_{\mathcal{A}}$. We say that m *encodes* $(G, \mathbf{d})$, written $m \rhd (G, \mathbf{d})$ iff: (i) for all $x \in \mathcal{X}$: $m(x, \mathbf{d}(x)) = 1$, (ii) for all $x \in \mathcal{X}$: for all $d \in \mathbb{D} \setminus \{\mathbf{d}(x)\}$: $m(x, d) = 0$ and (iii) for all $s \in S$ $m(s) = \mathsf{Parikh}(G)(s)$. Then:

Lemma 2 *Let $\mathcal{A}$ be a concurrent asynchronous* QDAS *with set of variables $\mathcal{X}$ and set of locations S, and let $N_{\mathcal{A}}$ be its associated* PN. *Then, for all $(G, \mathbf{d}) \in Reach(\mathcal{A})$ there is $m \in Reach(N_{\mathcal{A}})$ s.t. $m \rhd (G, \mathbf{d})$ and for all $m \in Reach(N_{\mathcal{A}})$, there is $(G, \mathbf{d}) \in Reach(\mathcal{A})$ s.t. $m \rhd (G, \mathbf{d})$.*

Proof. We prove the two statements separately.

Let $(G, \mathbf{d})$ be a configuration in $Reach(\mathcal{A}_N)$, and let

$$(G_0, \mathbf{d}_0)a_0(G_1, \mathbf{d}_1)a_1 \cdots a_{n-1}(G_n, \mathbf{d}_n)$$

be a run s.t. $(G, \mathbf{d}) = (G_n, \mathbf{d}_n)$. Let us build, inductively, a run $m_0 m_1 \cdots m_k$ of $N_{\mathcal{A}}$ s.t. $m_k \rhd (G, \mathbf{d})$. The induction is on the length n of the QDAS run.

Base case $n = 0$. It is easy to check that $m_0 \rhd (G_0, \mathbf{d}_0)$.

Inductive case $n = \ell$. Let us assume that $m_0 m_1 \cdots m_j$ is a run of $N_{\mathcal{A}}$ s.t. $m_j \rhd (G_{\ell-1}, \mathbf{d}_{\ell-1})$, and let us show how to complete it, if needed. We consider several case depending on a_{n-1}. In the case where $a_{n-1} = \varepsilon$ and the scheduler action consists in dequeueing a block from a queue, we have $\mathsf{Parikh}(G_{\ell-1}) = \mathsf{Parikh}(G_\ell)$ and $\mathbf{d}_\ell = \mathbf{d}_{\ell-1}$. By induction hypothesis $m_j \rhd (G_{\ell-1}, \mathbf{d}_{\ell-1})$, hence $m_j \rhd (G_\ell, \mathbf{d}_\ell)$, and we do not add elements to the run built so far. In the case where $a_{\ell-1} = \mathtt{dispatch_a}(\gamma, q)$, we assume $(s, a_{\ell-1}, s') \in \Delta$ is the corresponding LTS transition. Clearly, $\mathsf{Parikh}(G_\ell)(s') = \mathsf{Parikh}(G_{\ell-1})(s') + 1$, $\mathsf{Parikh}(G_\ell)(s) = \mathsf{Parikh}(G_{\ell-1})(s) - 1$, $\mathsf{Parikh}(G_\ell)(s_\gamma^0) = \mathsf{Parikh}(G_{\ell-1})(s_\gamma^0) + 1$ and for all other location s: $\mathsf{Parikh}(G_\ell)(s) = \mathsf{Parikh}(G_{\ell-1})(s)$. It is easy to check that the PN transition t s.t. $I(t)(p) = 1$ iff $p = s$ and $O(t)(p) = 1$ iff $p \in \{s', s_\gamma^0\}$ is fireable from m_j (as $m_j \rhd (G_{\ell-1}, \mathbf{d}_{\ell-1})$ by induction hypothesis) and yields the same effect, i.e. the marking m with $m_j \xrightarrow{t} m$ is s.t. $m \rhd (G_\ell, \mathbf{d}_\ell)$. All the other cases (test, assignment and task termination) are treated similarly.

Now, let $m_0 m_1 \cdots m_n$ be a run of $N_{\mathcal{A}}$ and let us build, inductively, a run

$$(G_0, \mathbf{d}_0)a_0(G_1, \mathbf{d}_1)a_1 \cdots a_{k-1}(G_k, \mathbf{d}_k)$$

s.t. $m_n \rhd (G_k, \mathbf{d}_k)$ *and* all the queues are empty in G_n. The induction is on the length n of the PN run.

Base case $n = 0$. It is easy to check that $m_0 \rhd (G_0, \mathbf{d}_0)$.

Inductive case $n = \ell$**.** Let us assume that $(G_0, \mathbf{d}_0)a_0 \cdots a_{j-1}(G_j, \mathbf{d}_j)$ is a run of $\mathcal{A}$ s.t. $m_{\ell-1} \rhd (G_j, \mathbf{d}_j)$ and all the queues are empty in G_j. Let t be the PN transition s.t. $m_{\ell-1} \xrightarrow{t} m_\ell$ and let us show how we can extend the run of $\mathcal{A}$. We consider several cases. If t is a transition that corresponds to an asynchronous dispatch, then there are s, s', γ and q s.t. $I_t(p) = 1$ iff $p = s$ and $O_t(p) = 1$ iff $p \in \{s', s_\gamma^0\}$. By definition of $N_{\mathcal{A}}$, there is a transition $(s, \texttt{dispatch}_\texttt{a}(\gamma, q), s')$ in $\mathcal{A}$. Moreover, $m_{\ell-1}(s) \geq 1$, since t is fireable from $m_{\ell-1}$. As $m_{\ell-1} \rhd (G_j, \mathbf{d}_j)$, the $(s, \texttt{dispatch}_\texttt{a}(\gamma, q), s')$ is fireable from $(G_j, \mathbf{d}_j)$, and leads to a configuration $(G_{j+1}, \mathbf{d}_{j+1})$, where a γ block has been enqueued in q, hence $\mathbf{d}_{j+1} = \mathbf{d}_j$, $\mathsf{Parikh}(G_{j+1})(s) = \mathsf{Parikh}(G_j)(s) - 1$, $\mathsf{Parikh}(G_{j+1})(s') = \mathsf{Parikh}(G_j)(s') + 1$, $\mathsf{Parikh}(G_{j+1})(s_\gamma^0) = \mathsf{Parikh}(G_j)(s_\gamma^0) + 1$ and for all other state s'': $\mathsf{Parikh}(G_{j+1})(s'') = \mathsf{Parikh}(G_j)(s'')$. It is easy to check that $m_\ell \rhd (G_{j+1}, \mathbf{d}_{j+1})$, however, queue q contains a call to γ in G_{j+1} and is thus the only non-empty queue in this CTG. Thus, from $(G_{j+1}, \mathbf{d}_{j+1})$, we execute the scheduler action that dequeues from q. This has no effect on the Parikh image of the CTG. Thus, we reach $(G_{j+2}, \mathbf{d}_{j+2})$ s.t. $\mathbf{d}_{j+1} = \mathbf{d}_{j+2}$, $\mathsf{Parikh}(G_{j+1}) = \mathsf{Parikh}(G_{j+2})$, hence $m\ell \rhd (G_{j+2}, \mathbf{d}_{j+2})$ too, and all the queues are empty in G_{j+2}, which concludes the induction step. All the other cases are treated similarly. $\qquad\square$

We can now prove Proposition 5: *Proof.* It is easy to check that the construction of $N_{\mathcal{A}}$, as described above, is polynomial. Then, assume f is Parikh coverable in $\mathcal{A}$, i.e. there is $(G, \mathbf{d}) \in Reach\,\mathcal{A}$ s.t. $f \preceq \mathsf{Parikh}(G)$. By Lemma 2, there is $m' \in Reach(N_{\mathcal{A}})$ s.t. $m' \rhd (G, \mathbf{d})$. Hence, for all $s \in S$: $m'(s) = \mathsf{Parikh}(G)(s)$. So, for all $s \in S$: $m(s) = f(s) \leq \mathsf{Parikh}(G)(s) = m'(s)$. Hence, $m \preceq m'$ (as $m(p) = 0$ for all $p \notin S$). Since $m' \in Reach(N_{\mathcal{A}})$, we conclude that $m \in Cover(N_{\mathcal{A}})$. On the other hand, assume $m \in Cover(N_{\mathcal{A}})$, with $m(p) = 0$ for all $p \notin S$, and let f be s.t. for all $s \in S$: $f(s) = m(s)$. Since $m \in Cover(N_{\mathcal{A}})$, there is $m' \in Reach(N_{\mathcal{A}})$ s.t. $m \preceq m'$. By Lemma 2, there is $(G, \mathbf{d}) \in Reach(\mathcal{A})$ s.t. $m' \rhd (G, \mathbf{d})$. Thus, by definition of $\rhd$, for all $s \in S$: $m'(s) = \mathsf{Parikh}(G)(s)$. Thus, since $m \preceq m'$ and by definition of f, we conclude that for all $s \in S$: $f(s) = m(s) \leq m'(s) = \mathsf{Parikh}(G)(s)$. Hence, f is Parikh-coverable in $\mathcal{A}$. $\qquad\square$

Proposition 6 *For all* PN *N, we can build, in polynomial time, a concurrent asynchronous* QDAS *$\mathcal{A}_N$ s.t. $m \in Cover(N)$ iff there exists $(G, \mathbf{d}) \in Reach(\mathcal{A}_N)$ with $G \rhd m$.*

The proof of Proposition 6 is split into two lemmata, given hereunder. They rely on an alternate characterization of $Cover(N)$. That is, $m \in Cover(N)$ iff m is reachable by a so-called *lossy* run of N, i.e. a sequence of markings $m'_0 m'_1 \cdots m'_n$ s.t. $m'_0 \preceq m_0$ and for all $0 \leq i \leq n - 1$: there is $\overline{m}_{i+1}$ and a transition t_i s.t. $m'_i \xrightarrow{t_i} \overline{m}_{i+1}$ and $m'_{i+1} \preceq \overline{m}_{i+1}$. Intuitively, a lossy run corresponds to firing a transition of the PN, and then spontaneously losing some tokens. The proof of these lemmata also assumes that each $p \in P$, the LTS $\mathcal{TS}_p = \langle \{s_P^0, s_p^{mid}, s_p^{fin}\}, s_p^0, \Sigma, \Rightarrow \rangle$ is as depicted in Fig. 7.

Lemma 3 *Let $N = \langle P, T, m_0 \rangle$ be a* PN*, and let $\mathcal{A}_N = \langle CQID, \emptyset, \Gamma, \texttt{main}, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ be its corresponding* QDAS*.* **If** *$m \in Cover(N)$* **then** *there exists $(G, \mathbf{d}) \in Reach(\mathcal{A}_N)$ s.t. $G \rhd m$.*
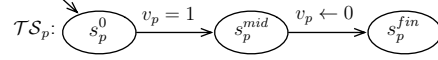
Figure 7: The LTS of bloc `p`

Proof. Let m be a marking from *Cover*(N). and let $m'_0 m'_1 \cdots m'_n$ be a lossy PN run s.t. $m = m_n$. The proof is by induction on the length of the run. More precisely, we show that, for all $0 \leq i \leq n$, there is a reachable configuration $(G_i, \mathbf{d}_i) \in \textit{Reach}(\mathcal{A}_N)$ s.t.: for all $p \in P$: $\mathbf{d}_i(v_p) = 0$, $G_i = \langle V^i, E^i, \lambda^i, \textit{queue}^i, \textit{state}^i \rangle$, $G_i \rhd m$, $m \preceq m'_i$ and $E^i = \emptyset$.

Base case: m'_0. Let us consider the run of $\mathcal{A}_N$ that consists in: (a) executing block `main` up to line 8, then (b) emptying the queue C. The execution of (a) has the effect that: (i) all v_p variables are initialized to 0 and keep this value, (ii) for all place p: *at most $m_0(p)$ copies* of block `p` are asynchronously dispatched in queue C and (iii) one copy of block `trans` is dispatched in C. Then, the execution of (b) creates one running task for each block that is present in C. Thus, the execution of (a) followed by (b) reaches a configuration $(G_0, \mathbf{d}_0)$ with $G_0 = \langle V^0 = V^0_T \uplus V^0_C, E^0, \lambda^0, \textit{queue}^0, \textit{state}^0 \rangle$ s.t. $V^0_C = \emptyset$ (the queue has been emptied), for all p: $|\{v \in V^0_T \,|\, \lambda(v) = \mathtt{p}\}| = m_0(p)$, $|\{v \in V^0_T \,|\, \lambda(v) = \mathtt{trans}\}| = 1$ and $E^0 = \emptyset$ (the queue is empty and all the calls are asynchronous). Moreover, *state* is such that each task running a `p` block is still in its initial state s^0_p, hence $G_0 \rhd m_0$. Similarly, the task running the `trans()` block is about to enter the `while` loop at line 14. Finally, as the variables have been initialized to 0 and not modified, we have $\mathbf{d}_0(v_p) = 0$ for all $p \in P$.

Inductive case: m_i Let us assume there exist $(G_{i-1}, \mathbf{d}_{i-1}) \in \textit{Reach}(\mathcal{A}_N)$ that respects all the conditions given at the beginning of the proof (in particular $G_{i-1} \rhd m_{i-1}$). Let t_i and $\overline{m}_i$ be the PN transition and marking s.t. $m_{i-1} \xrightarrow{t_i} \overline{m}_i$ and $m_i \preceq \overline{m}_i$ and let us show that $\mathcal{A}_N$ can simulate it. This is achieved by the following sequence of actions in $\mathcal{A}_N$. First, the block executing `trans` enters the `while` loop at line 14 and selects t_i as transition t. Then, it sets all the variables v_p s.t. $I_{t_i}(p) = 1$ to 1. Thus, at that point v_p contains 1 iff $I_{t_i}(p) = 1$, since all v_p variables were equal to 0 by induction hypothesis. Then, the task executing `trans` is blocked as it need to wait up to the point were all v_p are equal to 0. Since $G_{i-1} \rhd m_{i-1}$ by induction hypothesis, we know that there are, in G_{i-1}, $m_{i-1}(p)$ tasks executing block `p`, for all $p \in P$. However, t_i is fireable from m_{i-1}, and a loss of $\overline{m}_i - m_i$ token is still possible after the firing. Hence, $m_{i-1}(p) \geq (I_{t_i}(p) + \overline{m}_i(p) - m_i(p))$ for all p. Thus, for all p, there is at least $(I_{t_i}(p) + \overline{m}_i(p) - m_i(p))$ tasks executing `p` in G_{i-1}. Thus, we complete the run of $\mathcal{A}_N$ by letting, for all p, $(I_{t_i}(p) + \overline{m}_i(p) - m_i(p))$ `p` task execute lines 11 in turn one after the other. Then, letting them all execute line 12, and reach their final state (Remark that all the `p` task must first execute line 11 before one of them can execute line 12, as this sets v_p to 0 and would prevent other tasks to execute line 11). This is possible because none of those tasks are blocked, since the CTG contains no edge, by induction hypothesis. At that point, $\mathcal{A}_N$ has reached a configuration $(G', \mathbf{d}')$ s.t. $\mathbf{d}'(v_p) = 0$ for all $p \in P$ (by line 12) and where $G' \rhd m_{i-1} - (I_{t_i} + \overline{m}_i - m_i)$. Moreover, G' still respects all the other hypothesis as no new dispatch have been performed. Then, the simulation of t_i proceeds by letting the `trans` task finish the current iteration of the main `while`

24

loop. This consists in executing the `for` loop of line 19, which dispatches one `p` block in C iff $O_{t_i}(p) = 1$, i.e., the effect of t_i is to add a token to p. Finally, the scheduler empties queue C and creates tasks for all the blocks that have just been added to C. It also kills all the `p` tasks that have reached their final state. As a consequence, the configuration that is reached is $(G_i, \mathbf{d}_i)$, where $G_i \rhd m_{i-1} - (I_{t_i} + \overline{m}_i - m_i) + O_{t_i} = (m_{i-1} - I_{t_i} + O_{t_i}) - \overline{m}_i + m_i = \overline{m}_i - \overline{m}_i + m_i = m_i$ and $\mathbf{d}_i$ is s.t. $\mathbf{d}_i(v_p) = 0$ for all $p \in P$. Moreover, since the queue has been emptied by the scheduler, G_i contains only task nodes and no edge, as all the calls are asynchronous. The task executing `trans` is still active and at line 14, and all the `p` tasks are in their initial state. $\square$

Lemma 4 *Let* $N = \langle P, T, m_0 \rangle$ *be a* PN*, and let* $\mathcal{A}_N = \langle CQID, \emptyset, \Gamma, \texttt{main}, \mathcal{X}, \Sigma, (\mathcal{TS}_\gamma)_{\gamma \in \Gamma} \rangle$ *be its corresponding* QDAS*.* **If** *there are* $(G, \mathbf{d}) \in Reach(\mathcal{A}_N)$ *and* m *s.t.* $G \rhd m$ **then** $m(G) \in Cover(N)$.

Proof. For a CTG G of $\mathcal{A}_N$ with set of vertices V, we denote by $M(G)$ the marking of N s.t. for all $p \in P$: $M(G)(p) = |\{v \in V \mid state(v) = s_p^0\}|$. Thus, in the case where G encodes a configuration s.t. `trans` is at line 14, `main` is at line 8, and all the `p` blocks are in their initial state, then $G \rhd M(G)$.

In order to establish the lemma, we prove a stronger statement: every time we reach, along a run, a configuration $(G, \mathbf{d})$ s.t. `trans` is at line 14, then $M(G) \in Cover(N)$. Formally, let $\rho = (G_0, \mathbf{d}_0) a_0 (G_1, \mathbf{d}_1) a_1 (G_2, \mathbf{d}_2) \cdots (G_n, \mathbf{d}_n)$ be a run of $\mathcal{A}_N$, where, for all $0, \leq i \leq n$: $G_i = \langle V_i, E_i, \lambda_i, queue_i, state_i \rangle$. Let $\pi : \{0, \ldots, k\} \to \{0, \ldots, n\}$ be the monotonically increasing function s.t. $k \leq n$ and for all $0 \leq j \leq n$: there exists $v \in V_j$ with $state_i(v) = s_{\texttt{trans}}^{14}$ iff there is $0 \leq \ell \leq k$ with $k = \pi(\ell)$. That is the sequence $\pi(1), \pi(2), \ldots, \pi(k)$ identifies the indexes of all the configurations of the run where `trans` is at line 14. Let us show, by induction on i that all the $M(G_{\pi(i)})$'s are reachable *in the lossy semantics* of N.

Base case $i = 0$ Let us show that $M(G_{\pi(0)}) = m_0$, i.e., that the first time `trans` reaches line 14, $M(G_{\pi(0)})$ is the initial marking of N. Observe that the prefix of the run must have the following form. Initially, only the `main` block is executing: it first sets all the variables v_p to 0, then dispatches asynchronously at most $m_0(p)$ calls to each `p` block (for all $p \in P$), then finally dispatches an asynchronous call to `trans` and reaches line 8. Along this execution, the scheduler might decide to pick up some `p` blocks from C. However, as long as the scheduler has not scheduled the call to `trans`, the CTG met along the run do not encode any marking, by definition of $\rhd$. When the scheduler starts a task to run the `trans` block, we thus reach a configuration $(G, \mathbf{d})$ where: (i) the queue C is empty, as dequeueing the `trans` block is possible only if all the `p` blocks have been dequeued, and no other dispatch has been performed; (ii) all the `p` tasks are blocked in their initial state as $\mathbf{d}(v_p) = 0$ for all $p \in P$; and (iii) `main` is still blocked in the infinite loop at line 8. Since the scheduler has just dequeued `trans` from C, G is necessarily the first CTG to encode a marking, so $G = G_{\pi(0)}$. Moreover, by the loop at line 4, it is clear that $G \rhd m$ with $m \preceq m_0$.

Inductive case $i = \ell \geq 1$ The induction hypothesis is that $M(G_{\pi(i-1)}) \in Cover(N)$. Let us consider the $\rho' = (G_{\pi(\ell-1)}, \mathbf{d}_{\pi(\ell-1)}) \cdots (G_{\pi(\ell)}, \mathbf{d}_{\pi(\ell)})$, i.e. the portion of ρ that allows to reach $(G_{\pi(\ell)}, \mathbf{d}_{\pi(\ell)})$ from $(G_{\pi(\ell-1)}, \mathbf{d}_{\pi(\ell-1)})$. We consider two cases:

1. Either **trans** has not performed an iteration of its main `while` loop along ρ'. In

this case, the only actions that can occur along ρ' are scheduler actions consisting in dequeueing p blocks or the termination of some p tasks that where still in state s_p^{mid}. In both cases, this does not modify the value of $M(G)$, so $M(G_{\pi(i)}) = M(G_{\pi(i-1)}) \in \textit{Cover}(N)$.

2. Or $\mathtt{trans}$ has performed a complete iteration of its main $\mathtt{while}$ loop possibly interleaved with the dequeue of p blocks and the termination of p tasks. Since the dequeues and terminations have no influence on the value of $M(G)$ as argued above, let us focus on the effect of executing one iteration of the $\mathtt{while}$ loop. The iteration first selects a PN transition t and sets all the variables v_p s.t. $I_t(p) = 1$ to 1. The reached configuration is then $(G, \mathbf{d})$ where $M(G) = M(G_{\pi(i-1)})$, as these operations do not manipulate p blocks or tasks. Then, $\mathtt{trans}$ is blocked by the test at line 18. As only p blocks can set v_p variables to 0, we are sure that, when $\mathtt{trans}$ reaches line 19, *at least* $I_t(p)$ p blocks have left their initial state, for all $p \in P$. Thus, when $\mathtt{trans}$ is at line 19, the configuration is $(G', \mathbf{d}')$, where for all $p \in P$: $M(G')(p) \leq M(G)(p) - I_t(p) = M(G_{\pi(i-1)}) - I_t(p)$. Afterwards, $\mathtt{trans}$ terminates the iteration of the $\mathtt{while}$ loop by dispatching $O_t(p)$ p blocks for all $p \in P$, and reaches line 14, which finishes ρ'. Hence, we reach $(G_{\pi(i)}, \mathbf{d}_{\pi(i)})$, where for all $p \in P$: $M(G_{\pi(i)})(p) \leq M(G_{\pi(i-1)}) - I_t(p) + O_t(p)$. Since $M(G_{\pi(i-1)}) \in \textit{Cover}(N)$ by induction hypothesis, we conclude that $M(G_{\pi(i)}) \in \textit{Cover}(N)$ too. □

B.3 Asynchronous Serial QDAS

The proof of Theorem 4 relies on the next Lemma, that formalizes the relationship between reachable configurations of $\mathcal{A}_F$ and reachable configurations of F.

For all $\gamma \in \Gamma$, we denote by s_γ^ℓ the location of $\mathcal{TS}_\gamma$ that corresponds to line ℓ in Fig. 5. Then, we say that a configuration $(G, \mathbf{d})$ of $\mathcal{A}_F$ encodes a configuration (s, w) of F, written $(G, \mathbf{d}) \rhd (s, w)$ iff: (i) $s = \mathbf{d}(\mathtt{state})$, (ii) G is of either shapes in Fig. 5 with $w = m_0 m_1 \cdots m_n$, (iii) $\mathsf{Parikh}(G)(s_{\mathtt{main}}^8) = 1$ and (iv) there exists $m \in M \cup \{\varepsilon\}$ s.t. $\mathsf{Parikh}(G)(s_{\mathtt{m}}^{12}) = 1$. That is, s and w are encoded as described above, $\mathtt{main}$ is at line 8, and the running $\mathtt{m}$ block is at line 12. Then:

Lemma 5 *Let F be a FIFO system, let c be a configuration of F, and let $\mathcal{A}_F$ be its associated* QDAS*. For all run $(s_0, w_0)(s_1, w_1) \cdots (s_n, w_n)$ of F s.t. for all $0 \leq i < n$: $s_i \neq c$, there exists $(G, \mathbf{d}) \in \textit{Reach}(\mathcal{A}_F)$ s.t. $(G, \mathbf{d}) \rhd (s_n, w_n)$. Moreover, for all $(G, \mathbf{d}) \in \textit{Reach}(\mathcal{A}_F)$ and for all configuration (s, w) of F: $(G, \mathbf{d}) \rhd (s, w)$ implies $(s, w) \in \textit{Reach}(F)$*

Proof. First, we consider a run $(s_0, w_0)(s_1, w_1) \cdots (s_n, w_n)$ of F s.t. for all $0 \leq i < n$: $s_i \neq c$, and build a run $(G_0, \mathbf{d}_0) a_0 (G_1, \mathbf{d}_1) a_1 \cdots a_{k-1} (G_k, \mathbf{d}_k)$ of $\mathcal{A}_F$ s.t. $(G_k, \mathbf{d}_k) \rhd (s_n, w_n)$, by induction on the length of F's run.

Base case $n = 0$**:** Consider the run of $\mathcal{A}_F$ that consists in executing lines 5, 6, 7 of $\mathtt{main}$ (which sets the $\mathtt{head}$ variable to ε), then dequeueing the ε block from the queue, then executing lines 10 and 11 of ε. Remark that the test at line 10 is not satisfied, as $\mathtt{head} = \varepsilon$, and that the queue is now empty. Clearly, the resulting configuration $(G, \mathbf{d}) \rhd (s_F^0, w_0)$ as $w_0 = \varepsilon$.

Inductive case $n = \ell$. Let us assume that there is a reachable configuration $(G, \mathbf{d})$ of $\mathcal{A}_F$ s.t. $(G, \mathbf{d}) \rhd (s_{\ell-1}, w_{\ell-1})$, and let us build a sequence of $\mathcal{A}_F$ transitions that is fireable from $(G, \mathbf{d})$ and reaches a configuration encoding (s_ℓ, w_ℓ). In $(G, \mathbf{d})$, there is, by definition of $\rhd$, a task running a b block, for $b \in M \cup \{\varepsilon\}$, that is at line 12. Moreover, $\mathbf{d}(\texttt{state}) = s_{\ell-1}$. Let δ be the transition of F s.t. $(s_{\ell-1}, w_{\ell-1}) \xrightarrow{\delta} (s_\ell, w_\ell)$. By hypothesis, $s_{\ell-1} \neq c$, hence, we let b execute line 12; select $\delta = (s_{\ell-1}, a, s_\ell)$ at line 13; execute line 14, where the condition of the if is not satisfied as $s = s_{\ell-1} = \texttt{state}$; and execute line 16, which reaches a configuration $(G', \mathbf{d}')$ where $\mathbf{d}'(\texttt{state}) = s_\ell$. We consider three cases to complete the simulation of δ in $\mathcal{A}_F$. If $a = !n$, the b task performs an asynchronous dispatch of n to $\mathtt{q}$, and jumps to line 11, then 12. Clearly, the resulting configuration $(G'', \mathbf{d}'')$ is s.t. $(G'', \mathbf{d}'') \rhd (s_\ell, w_\ell)$ (in particular, the dispatch has correctly updated the content of the queue). If $a = \varepsilon$, the b tasks jumps directly to line 11, then to line 12. Again, the resulting configuration $(G'', \mathbf{d}'')$ is s.t. $(G'', \mathbf{d}'') \rhd (s_\ell, w_\ell)$, as the content of the queue has not been modified. Finally, if $a = !n$, the running b block sets $\texttt{head}$ to n and terminates. Let $(G'', \mathbf{d}'')$ be the $\mathcal{A}_F$ configuration reached at that point. As δ is fireable from $(s_{\ell-1}, w_{\ell-1})$ in F, since $(G, \mathbf{d}) \rhd (s_{\ell-1}, w_{\ell-1})$, and as the content of the queue has not been modified since then, the head of $\mathtt{q}$ is necessarily an n block in G''. Moreover, $\mathbf{d}''(\texttt{head}) = n$ and $\mathbf{d}''(\texttt{state}) = s_\ell$. Thus, we let the scheduler dequeue this n block, and we let the task running it execute line 10 (where the condition of the if is not satisfied), then line 11. Clearly, the resulting configuration encodes (s_ℓ, w_ℓ).

Now, let $\rho = (G_0, \mathbf{d}_0)a_0(G_1, \mathbf{d}_1)a_1 \cdots a_{n-1}(G_n, \mathbf{d}_n)$ be a run of $\mathcal{A}_F$ s.t. there is (s, w) with $(G_n, \mathbf{d}_n) \rhd (s, w)$, and let us build, by induction on the length of this run, a run $(s_F^0, w_0)(s_1, w_1) \cdots (s_k, w_k)$ a run of F s.t. $(s_k, w_k) = (s, w)$.

Let $K = |\{(G_i, \mathbf{d}_i) \mid \mathsf{Parikh}(G_i)(s_\mathtt{m}^{12}) = 1 \text{ for } m \in M \cup \{\varepsilon\}\}|$, i.e., K is the number of times an m block reaches line 12 along ρ. Let us consider the increasing monotonic function $\rho : \{1, \ldots, K\} \to \{0, \ldots, n\}$ s.t. for all $0 \leq i \leq n$: there exists $m \in M \cup \{\varepsilon\}$ s.t. $\mathsf{Parikh}(G_i)(s_\mathtt{m}^{12}) = 1$ iff there is $1 \leq j \leq K$ s.t. $G_i = \rho(j)$, that is, $\rho(i)$ is the index, in ρ of the ith time a configuration is reached where an $\mathtt{m}$ block is at line 12. Clearly, by definition of $\rhd$ only the $(G_{\rho(j)}, \mathbf{d}_{\rho(j)})$ configurations (for $1 \leq j \leq K$) can encode a configuration of F, as no m block is at line 12 in the other configurations of ρ. So, it is sufficient to show that all those $(G_{\rho(j)}, \mathbf{d}_{\rho(j)})$ configurations encode a reachable configuration of F. We proceed by induction on j, and show that: for all $1 \leq j \leq K$: $(G_{\rho(j)}, \mathbf{d}_{\rho(j)})$ encodes a reachable configuration of F and $G_{\rho(j)}$ contains exactly one m task (for $m \in M \cup \{e\}$), that has been dequeued from $\mathtt{q}$.

Base case $j = 0$: Observe that the subrun $(G_0, \mathbf{d}_0)a_0 \cdots a_{\rho(1)-1}(G_{\rho(1)}, \mathbf{d}_{\rho(1)})$ is necessarily an initialization phase where $\texttt{main}$ sets $\texttt{state}$ to s_F^0, $\texttt{head}$ to ε, dispatches an ε block, and reaches line 8, where it will stay forever. Then, the scheduler dequeues the ε block, which empties the queue. The ε task then traverses line 10 (as $\texttt{head} = \varepsilon$) and 11 and reaches line 12. So, clearly $(G_{\rho(0)}, \mathbf{d}_{\rho(0)}) \rhd (s_F^0, \varepsilon)$ and contains exactly one m task (for $m \in M \cup \{e\}$), that has been dequeued from $\mathtt{q}$.

Inductive case $j = \ell$: Let us assume that $(G_{\rho(\ell-1)}, \mathbf{d}_{\rho(\ell-1)})$ encodes a reachable configuration $(s_{\ell-1}, w_{\ell-1})$ of F. We consider several cases. If $(G_{\rho(\ell-1)}, \mathbf{d}_{\rho(\ell-1)}) = (G_{\rho(\ell)}, \mathbf{d}_{\rho(\ell)})$ we are done. Otherwise, we have necessarily performed one iteration (possibly interrupted at line 12, 14 or 19) of the while loop starting at line 11, between

27

$(G_{\rho(\ell-1)}, \mathbf{d}_{\rho(\ell-1)})$ and $(G_{\rho(\ell)}, \mathbf{d}_{\rho(\ell)})$, as, by induction hypothesis, $G_{\rho(\ell-1)}$ contains exactly one m task (with $m \in M \cup \{\varepsilon\}$) that blocks q, and main can only loop at line 8, which does not modify the current configuration. Then, observe that the conditions of the if at lines 12 and 14 were necessarily false during the iteration. Otherwise, m would have reached line 21, from which it cannot escape. From that point, no configuration is reachable where an m block is at line 12 , and $(G_{\rho(ell)}, \mathbf{d}_{\rho(\ell)})$ cannot exist. Thus, we consider three cases:

- If we have entered the if at line 16 during the iteration, then a transition of the form $(s, !n, s')$ has been guessed, with state $= s$ and a dispatch of n has been performed into q. As $(G_{\rho(\ell-1)}, \mathbf{d}_{\rho(\ell-1)}) \triangleright (s_{\ell-1}, w_{\ell-1})$ by induction hypothesis, $s_{\ell-1} = s$, and thus $(s, !n, s')$ is fireable from $(s_{\ell-1}, w_{\ell-1})$ and reaches $(s', n \cdot w_{\ell-1})$. Clearly, this configuration is encoded by $(G_{\rho(\ell)}, \mathbf{d}_{\rho(\ell)})$.

- If we have entered the else if at line 17 during the iteration, then a transition of the form $(s, ?n, s')$ has been guessed, with state $= s$, head has been set to n, the current m block has been terminated, a new block m' has been dequeued by the scheduler (as there is necessarily a running m block in $G_{\rho(\ell)}$). Moreover $m' = n$, because m' has to be at line 12 in $G_{\rho(\ell)}$, so the test of line 10 had to be false to allow m' to reach line 12. As $(G_{\rho(\ell-1)}, \mathbf{d}_{\rho(\ell-1)}) \triangleright (s_{\ell-1}, w_{\ell-1})$ by induction hypothesis, $s_{\ell-1} = s$. As a dequeue of a block $m' = n$ has been performed, $w_{\ell-1}$ is of the form $w \cdot n$. Thus, $(s, ?m, s')$ is fireable from $(s_{\ell-1}, w_{\ell-1})$ and reaches (s', w). Clearly, this configuration is encoded by $(G_{\rho(\ell)}, \mathbf{d}_{\rho(\ell)})$.

- Finally, if neither the if nor the else if have been entered during the iteration, then a transition of the form (s, ε, s') has been guessed, with state $= s$. As $(G_{\rho(\ell-1)}, \mathbf{d}_{\rho(\ell-1)}) \triangleright (s_{\ell-1}, w_{\ell-1})$ by induction hypothesis, $s_{\ell-1} = s$, and thus (s, ε, s') is fireable from $(s_{\ell-1}, w_{\ell-1})$ and reaches $(s', w_{\ell-1})$. Clearly, this configuration is encoded by $(G_{\rho(\ell)}, \mathbf{d}_{\rho(\ell)})$. $\qquad\square$

We can now prove Theorem 4: *Proof.* Let F be a FIFO system, with set of messages M and associated serial asynchronous QDAS $\mathcal{A}_F$ and let c be a control location of F. For all $m \in M \cup \{\varepsilon\}$, let f_m be the Parikh image s.t. $f_m(s_{\texttt{main}}^{21}) = 1$ and $f_m(s) = 0$ for all $s \neq s_{\texttt{main}}^{21}$. Remark that there are only finitely many such f_m. Then, we show that c is reachable in F iff there exists $m \in M \cup \{\varepsilon\}$ s.t. f_m is Parikh-coverable in $\mathcal{A}_F$.

Assume c is reachable in F, and let (c, w) be a configuration in *Reach*(F). Without loss of generality, assume c is reachable by run that visits c only once. By Lemma 5, there is $(G, \mathbf{d}) \in$ *Reach*$(\mathcal{A}_F)$ s.t. $(G, \mathbf{d}) \triangleright (c, w)$. Hence, in $(G, \mathbf{d})$, there is a task running an m block (for $m \in M \cup \{\varepsilon\}$) that is at line 12, and $\mathbf{d}(\texttt{state}) = c$. Thus, m can execute one step and reach line 21, so f_m is Parikh coverable in $\mathcal{A}_F$.

For the reverse direction, assume there is $m \in M \cup \{\varepsilon\}$ that is Parikh-coverable in $\mathcal{A}_F$. Hence, there is $(G, \mathbf{d}) \in$ *Reach*$(\mathcal{A}_F)$ where a task running block m is at line 21. The only way for that block to reach line 21 is from line 12, with a valuation $\mathbf{d}'$ s.t. $\mathbf{d}'(\texttt{state}) = c$. Thus, there is, in *Reach*$(\mathcal{A}_F)$ a configuration $(G', \mathbf{d}')$ with $\mathbf{d}'(\texttt{state}) = c$, a task running an m block at line 12, and necessarily main at line 8 (otherwise, only main would be running). Hence, $(G', \mathbf{d}')$ is a reachable configuration

of $\mathcal{A}_F$ s.t. $(G', \mathbf{d}') \rhd (c, w)$ for some queue content w. Thus, by Lemma 5, $(c, w) \in$ *Reach*(F), and c is reachable in F.

We have thus reduced the control location reachability problem of FIFO systems to the Parikh coverability problem of serial asynchronous QDAS (using only one serial queue). The former is undecidable. Hence the theorem. □