Reducing Linearizability of Priority Queues and More Data Structures to State Reachability

Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Chao Wang

Institut de Recherche en Informatique Fondamentale,
Univ. Paris Diderot (Paris 7)

Abstract. I am abstract

1 Introduction

I am introduction

Related work Related work

2 Preliminaries

In this section, we introduce the notion of sequential executions, concurrent executions, histories and linearizability in [1,3]. Since *put* method of priority queue has two arguments, we slightly modified related definitions.

We fix several (possibly infinite) set $\mathbb{D}_1, \ldots$ of data values, and a finite set $\mathbb{M}$ of methods. We identify a subset $\mathbb{M}_{in} \subseteq \mathbb{M}$ of input methods in order to differentiated methods taking an argument (e.g., the *put* method which inserts a argument value into a priority queue) from the other methods (e.g., the *rm* method which doesn't take an argument, and returns the item with maximal priority of a queue). A method-event is composed of a method $m \in \mathbb{M}$ and several data value $x_1 \in \mathbb{D}_1, \ldots$, and is denoted $m(x_1, \ldots)$. We define the concatenation of method-event sequences $u \cdot v$ in the usual way, and ϵ denotes the empty sequence.

Definition 1. *A sequential execution is a sequence of method events.*

We also fix an arbitrary infinite set $\mathbb{O}$ of operation (identifiers). A call action is composed of a method $m \in \mathbb{M}$, several data value $x_1 \in \mathbb{D}_1, \ldots$, an operation $o \in \mathbb{O}$, and is denoted $cal_o(m, x_1, \ldots)$. Similarly, a return action is denoted $ret_o(m, x_1, \ldots)$. The operation o is used to match return actions to their call actions.

Definition 2. *A (concurrent) execution e is a sequence of call and return actions which satisfy a well-formedness property: every return has a call action before it in e, using the same tuple $m, o, x_1, \ldots$, and an operation o can be used only twice in e, once in a call action, and once in a return action.*

Example 1. $cal_{o_1}(put, a, 7) \cdot cal_{o_2}(put, b, 4) \cdot ret_{o_1}(put, a) \cdot ret_{o_2}(put, b)$ is a concurrent execution, while $cal_{o_1}(put, a, 7) \cdot cal_{o_2}(put, b, 4) \cdot ret_{o_1}(put, a) \cdot ret_{o_1}(put, b)$ and $cal_{o_1}(put, a, 7) \cdot ret_{o_1}(put, a) \cdot ret_{o_2}(put, b)$ are not.

Definition 3. *An implementation $\mathcal{I}$ is a set of concurrent executions.*

Implementations represent libraries whose methods are called by external programs. In the remainder of this work, we consider only completed executions, where each call action has a corresponding return action. This simplification is sound when implementation methods can always make progress in isolation [4]: formally, for any execution e with pending operations, there exists an execution e' obtained by extending e only with the return actions of the pending operations of e. Intuitively this means that methods can always return without any help from outside threads, avoiding deadlock.

We simplify reasoning on executions by abstracting them into histories.

Definition 4. *A history is a labeled partial order $(O, <_{hb}, l)$ with $O \in \mathbb{O}$ and l : maps each $o \in O$ into $\mathbb{M} \times \mathbb{D}_1$, or $\mathbb{M} \times \mathbb{D}_1 \times \mathbb{D}_2$, or $\ldots$.*

The order $<_{hb}$ is called the happens-before relation, and we say that o_1 happens before o_2 when $o_1 <_{hb} o_2$. Since histories arise from executions, their happens-before relations are interval orders [2]: for distinct o_1, o_2, o_3, o_4, if $o_1 <_{hb} o_2$ and $o_3 <_{hb} o_4$, then either $o_1 <_{hb} o_4$, or $o_3 <_{hb} o_2$. Intuitively, this comes from the fact that concurrent threads share a notion of global time.

The history of an execution e is defined as $O, <_{hb}, l$ where:

- O is the set of operations which appear in e,
- $o_1 <_{hb} o_2$, if the return action of o_1 is before the call action of o_2 in e,
- an operation o occurring in a call action $call_o(m, x)$ is labeled by $m(x)$, the case of multiple arguments are similar.

Example 2. The history of the execution $cal_{o_1}(put, a, 7) \cdot cal_{o_2}(put, b, 4) \cdot ret_{o_1}(put, a) \cdot ret_{o_2}(put, b)$ is $(\{o_1, o_2\}, <_{hb}, l)$ with $l(o_1) = put(a, 7)$, $l(o_2) = put(b, 4)$ and with $<_{hb}$ being the empty order relation, since o_1 and o_2 overlap.

Let $h = (O, <_{hb}, l)$ be a history and e a sequential execution of length n. We say that h is linearizable with respect to e, denoted $h \sqsubseteq e$, if there is a bijection $f : O \to \{1, \ldots, n\}$ s.t.

- if $o_1 <_{hb} o_2$, then $f(o_1) <_{hb} f(o_2)$,
- the method event at position $f(o)$ in e is $l(o)$.

Definition 5. *A history h is linearizable with respect to a set S of sequential executions, denoted $h \sqsubseteq S$, if there exists $e \in S$ such that $h \sqsubseteq e$.*

A set of histories H is linearizable with respect to S, denoted $H \sqsubseteq S$, if $h \sqsubseteq S$ for all $h \in H$. We extend these definitions to executions according to their histories. In that context, the set S is called a specification.

3 Inductive Rules of Extended Priority Queue

A priority queue contains two method: *put* and *rm*. A *put* method has two arguments, while the first argument is an item and the second argument is its priority. A *put* method is used to put an item into the priority queue with certain priority. Here we assume that the item is chosen from a specific (possibly infinite) data domain $\mathbb{D}$ and priority is chosen from a (possibly infinite) set $\mathbb{P}$. Moreover, we assume that there is a strict partial-order $<_{\mathbb{P}}$ among elements in $\mathbb{P}$. A *rm* method intends to remove the item with minimal priority (with respect to $<_{\mathbb{P}}$) in priority queue and then returns it. It works as follows:

- If the priority queue is empty, then *rm* returns *empty*.
- Else, *rm* returns a oldest element of one of minimal priorities. Formally, there are a set S of items in priority queue. S can be divided into several group, such that (1) items in each group have same priority, (2) the priorities of each two groups is incomparable and (3) no priority of items in S can be larger than items not in S. Each group is the set of items of some minimal priority. Then, *rm* returns an item of some group, and this item must be putted earliest in this group. However, the chosen of group is arbitrary.

We say that $put(a,p)$ matches $rm(b)$, if $a=b$. Our priority queue is an extension of common priority queue, where the priority is chosen from the set $\mathbb{N}$ of natural numbers, or from a set with total order. To distinguish our priority queue with common priority queue, we explicitly call our priority queue the extended priority queue.

Similar as [1], we use inductive rules to define the set of sequential executions of extended priority queue. Each rule is of the form $l_1 \cdot \ldots \cdot l_k \in \mathit{EPQ} \wedge \mathit{Guard}(l_1,\ldots,l_k,\mathit{itm},\mathit{pri}) \Rightarrow \mathit{Expr}(l_1,\ldots,l_k,\mathit{itm},\mathit{pri}) \in \mathit{EPQ}$. Here *EPQ* is the set of sequential executions of extended priority queue. Here *itm* and *pri* are two variables, and represent an arbitrary item and priority, respectively. $\mathit{Guard}(l_1,\ldots,l_k,\mathit{itm},\mathit{pri})$ is a conjunction of conditions with the following notations:

- Given sequential execution l, $\mathit{noRE}(l)$ is satisfied when each method event of l is not $\mathit{rm}(\mathit{empty})$.
- Given sequential execution l, $\mathit{LEI}(\mathit{pri},l)$ is satisfied, if for priority of every item of l, *pri* is either larger, or equal, or incomparable with it. Here L represents larger, E represents equal, and I represents incomparable. Similarly, we can define $\mathit{LI}(\mathit{pri},l)$. We use U to represent items of unmatched *put*, and $\mathit{LI\text{-}U}(\mathit{pri},l)$ is satisfied, if for priority of every item of unmatched *put* in l, *pri* is either larger, or incomparable with it. We use M to represent items of matched *put*, and define *LI-M* and *LEI-M* similarly.
  We use E' to emphasize that equal must holds somewhere. For example, $\mathit{LE'I}(\mathit{pri},l)$ holds, if (1) for priority of every item of l, *pri* is either larger, or equal, or incomparable with it, and (2) *pri* indeed equals priority of some items of l. *LE'I-M* is similarly defined.
- Given sequential execution l, its sub-sequence l' and a priority p, $\mathit{putInSeq}(l,l',p)$ is satisfied when all the *put* with priority p of l (if exists) is in l'.

- Given sequential execution l and priority p, *matched-C*(l,p) is satisfied, if (1) for each item a whose priority is comparable with p, if *put*$(a,\_)$ is in l, then *rm*(a) is in l, and (2) for each item a whose priority is comparable with p, if *rm*(a) is in l, then *put*$(a,\_)$ is in l. Similarly, we can define *matched-All*(l), where all items in l, instead of items with priority comparable with some priority in l, is considered and matched.

$Expr(l_1,\dots,l_k,itm,pri)$ is a expression $l'_1\cdot\ldots\cdot l'_m$, where each l'_i is chosen from (1) l_j for some j, (2) method event with item *itm* and priority *pri* and (3) method event *rm*(*empty*). Given a rule $R \equiv l_1\cdot\ldots\cdot l_k \in EPQ \wedge Guard(l_1,\dots,l_k,itm,pri) \Rightarrow Expr(l_1,\dots,l_k,itm,pri)\in EPQ$ and a sequential execution w, if $w=l'_1\cdot\ldots\cdot l'_k$, and $Guard(l'_1,\dots,l'_k,a,p)$ holds for some $a\in\mathbb{D}$ and $p\in\mathbb{P}$, then we use $w\xrightarrow{R}w'$ to denote that we can obtain w' from w according to rule R, where $w'=Expr(l'_1,\dots,l'_k,a,p)$. Let $[\![EPQ]\!]$ be the set of sequential executions w which can be derived from the empty word:

$$\epsilon = w_0 \xrightarrow{R_1} w_1 \ldots \xrightarrow{R_k} w$$

where each R_i is one rules of the extended priority queue. When clear from context, we abuse $[\![EPQ]\!]$ by *EPQ*.

Definition 6. *EPQ is defined by the following rules:*

- $EPQ_0 \equiv \epsilon \in EPQ$.
- $EPQ_1 \equiv (u\cdot v\cdot w\in EPQ)\wedge(noRE(u\cdot v\cdot w))\wedge(LEI(pri,u\cdot v\cdot w))\wedge(LI\text{-}U(pri,u\cdot v\cdot w))\wedge(matched\text{-}C(u\cdot v,pri))\wedge(putInSeq(u\cdot v\cdot w,u,pri))\Rightarrow(u\cdot put(itm,pri)\cdot v\cdot rm(itm)\cdot w\in EPQ)$.
- $EPQ_2 \equiv (u\cdot v\in EPQ)\wedge(noRE(u\cdot v))\wedge(LEI(pri,u\cdot v))\wedge(putInSeq(u\cdot v,u,pri))\Rightarrow(u\cdot put(itm,pri)\cdot v\in EPQ)$.
- $EPQ_3 \equiv (u\cdot v\in EPQ)\wedge(matched\text{-}All(u))\Rightarrow(u\cdot rm(empty)\cdot v\in EPQ)$.

Example 3. Given priorities p_1,p_2,p_3 with orders $p_1<_{\mathbb{P}}p_2$ and $p_1<_{\mathbb{P}}p_3$, one sequential execution of *EPQ* is generated as follows:

$\epsilon \xrightarrow{EPQ_1} put(a,p_1)\cdot rm(a)$
$\xrightarrow{EPQ_2} put(a,p_1)\cdot rm(a)\cdot put(b,p_1)$
$\xrightarrow{EPQ_1} put(c,p_2)\cdot put(a,p_1)\cdot rm(a)\cdot rm(c)\cdot put(b,p_1)$
$\xrightarrow{EPQ_2} put(c,p_2)\cdot put(a,p_1)\cdot rm(a)\cdot rm(c)\cdot put(d,p_2)\cdot put(b,p_1)$
$\xrightarrow{EPQ_1} put(c,p_2)\cdot put(a,p_1)\cdot rm(a)\cdot rm(c)\cdot put(d,p_2)\cdot put(e,p_3)\cdot rm(e)\cdot put(b,p_1)$
$\xrightarrow{EPQ_3} put(c,p_2)\cdot put(a,p_1)\cdot rm(a)\cdot rm(c)\cdot rm(empty)\cdot put(d,p_2)\cdot put(e,p_3)\cdot rm(e)\cdot put(b,p_1)$

To facilitate our proof of latter sections, we need to separate EPQ_1 into two cases: (1) no matched pair of *put* and *rm* in $u\cdot v\cdot w$ has priority *pri*, (2) some matched pair of *put* and *rm* in $u\cdot v\cdot w$ has priority *pri*. Therefore, we separate EPQ_1 into two rules:

- $EPQ_1^{>} \equiv (u\cdot v\cdot w\in EPQ)\wedge(noRE(u\cdot v\cdot w))\wedge(LI(pri,u\cdot v\cdot w))\wedge(LI\text{-}U(pri,u\cdot v\cdot w))\wedge(matched\text{-}C(u\cdot v,pri))\Rightarrow(u\cdot put(itm,pri)\cdot v\cdot rm(itm)\cdot w\in EPQ)$.

- $EPQ_1^{=} \equiv (u \cdot v \cdot w \in EPQ) \wedge (noRE(u \cdot v \cdot w)) \wedge (LE'I(pri, u \cdot v \cdot w)) \wedge (LI\text{-}U(pri, u \cdot v \cdot w)) \wedge (matched\text{-}C(u \cdot v, pri)) \wedge (putInSeq(u \cdot v \cdot w, u, pri)) \Rightarrow (u \cdot put(itm, pri) \cdot v \cdot rm(itm) \cdot w \in EPQ)$.

For EPQ_2, we also need to distinguish two cases: (1) no matched pair of *put* and *rm* in $u \cdot v$ has priority *pri*, (2) some matched pair of *put* and *rm* in $u \cdot v$ has priority *pri*. Therefore, we separate EPQ_2 into two rules:

- $EPQ_2^{>} \equiv (u \cdot v \in EPQ) \wedge (noRE(u \cdot v)) \wedge (LEI(pri, u \cdot v)) \wedge (LI\text{-}M(pri, u \cdot v)) \wedge (putInSeq(u \cdot v, u, pri)) \Rightarrow (u \cdot put(itm, pri) \cdot v \in EPQ)$.
- $EPQ_2^{=} \equiv (u \cdot v \in EPQ) \wedge (noRE(u \cdot v)) \wedge (LEI(pri, u \cdot v)) \wedge (LE'I\text{-}M(pri, u \cdot v)) \wedge (putInSeq(u \cdot v, u, pri)) \Rightarrow (u \cdot put(itm, pri) \cdot v \in EPQ)$.

To persuade readers that our rules is indeed the rules of extended priority queue, in Appendix A, we give a semantical version definition EPQ_s of extended priority queue, and shows that the language generated by our rules equals the set of traces of EPQ_s. To model the possible behaviors of extended priority queue, we model it as an labelled transition system (shortened as LTS) LTS_e. Each state of LTS_e is a function from $\mathbb{P}$ into sequences over $\mathbb{D}$, and represents a snapshot of contents of extended priority queue. EPQ_s is the set of traces of LTS_e. The definition of EPQ_s and the proof of the following lemma can be found in Appendix A.

Lemma 1. $EPQ = EPQ_s$.

Given a sequential execution $w \in EPQ$ and let P be the set of priorities in w. According to above example, w is generated from ϵ as follows: Let P be the set of priorities of w and p be one of minimal priority in P. Then we start to loop. In each round of the loop there are two possibilities: (1) If there are matched *put* and *rm* with priority p in w: Add pairs of matched *put* and *rm* with priority p by using one time of $EPQ_1^{>}$ and then (possibly) several times of $EPQ_1^{=}$, and then (possibly) add unmatched *put* with priority p by using $EPQ_2^{=}$, (2) If there are no matched *put* and *rm* with priority p in w: Add unmatched *put* with priority p by using $EPQ_2^{>}$ for several times. Then, let $P = P \setminus \{p\}$, and restart this loop, until $P = \emptyset$. Finally, add $rm(empty)$ by using EPQ_3 for several times. We can see that, the order for generating w may be not fixed, since some priorities are incomparable.

Thus, given w, we define $last(w)$ as the set of last possible rule to generate w according to the rules of extended priority queues:

- If w contains $rm(empty)$, then $last(w) = \{EPQ_3\}$.
- Else, if items of several unmatched *put* and matched *put* have a maximal priority of w, then $last(w)$ contains $EPQ_2^{=}$.
- Else, if items of only several unmatched *put* have a maximal priority of w, then $last(w)$ contains $EPQ_2^{>}$.
- Else, if items of only more than one matched *put* have a maximal priority of w, then $last(w)$ contains $EPQ_1^{=}$.
- Else, if items of only one matched *put* has a maximal priority of w, then $last(w)$ contains $EPQ_1^{>}$.

- Else ($w = \epsilon$), $\mathit{last}(w) = \{\mathit{EPQ}_0\}$.

Note that $\mathit{last}(w)$ may contains more than one rules. For example, given $w = \mathit{put}(c,p_2) \cdot \mathit{put}(a,p_1) \cdot \mathit{rm}(a) \cdot \mathit{rm}(c) \cdot \mathit{put}(d,p_2) \cdot \mathit{put}(e,p_3) \cdot \mathit{rm}(e) \cdot \mathit{put}(b,p_1)$ and orders $p_1 <_{\mathbb{P}} p_2$ and $p_1 <_{\mathbb{P}} p_3$, then $\mathit{last}(w) = \{\mathit{EPQ}_2^{=}, \mathit{EPQ}_1^{>}\}$. The notion of *last* can be extended into execution and histories: given a history h, $\mathit{last}(h) = \mathit{last}(u)$, where u is any sequential execution such that $h \sqsubseteq u$. When $\mathit{last}(e)$ contains only one rule R, we write $\mathit{last}(e) = R$ for simplicity.

4 Data-Independence of Extended Priority Queue

Data-independence [3] can be used to effectively handle unbounded data domain. In this section, we slightly modify the notion of data-independence in [3] and propose data-differentiated sequences and data-independence for extended priority queues.

Let $\_$ denote an element, of which the value is irrelevant. A sequential execution e of extended priority queue is said to be data-differentiated if, for all $d \in \mathbb{D}$, there is at most one method event $\mathit{put}(d, \_)$ in e. Note that a data-differentiated sequential execution e may contains more than one items with a same priority. The subset of data-differentiated sequential executions of a set S is denoted by $S_{\neq}$. The definition extends to (sets of) executions and histories. For instance, an execution is data-differentiated if, for all $d \in \mathbb{D}$, there is at most one $\mathit{cal}_{\_}(\mathit{put}, d, \_)$.

Example 4. $\mathit{cal}_{o_1}(\mathit{put}, a, 7) \cdot \mathit{ret}_{o_1}(\mathit{put}, a) \cdot \mathit{cal}_{o_2}(\mathit{put}, a, 8) \cdot \mathit{ret}_{o_2}(\mathit{put}, a)$ is not data-differentiated, since there are two *put* methods with the same item.

A renaming function r for extended priority queue is a function from $\mathbb{D}$ to $\mathbb{D}$. Given a sequential execution (resp., execution or history) e, we denote by $r(e)$ the sequential execution (resp., execution or history) obtained from e by replacing every item x by $r(x)$. Note that here the renaming functions rename only the items and keep the priorities unchanged. The intuitive explanation is that renaming items will not influence the executions of program, while renaming priorities may influence the executions of program.

Definition 7. *A set of sequential executions (resp., executions or histories) S is data-independent, if:*

- *for all $e \in S$, there exists $e' \in S'$, and a renaming function r, such that $e = r(e')$,*
- *for all $e \in S$ and for all renaming r, $r(e) \in S$.*

The following lemma states that, when checking that a data-independent implementation $\mathcal{I}$ is linearizable with respect to a data-independent specification, it is enough to do so for data-differentiated executions, similar as that in [5], where the notion of data-independence and differentiated in [3] is used. Thus, in the remainder of the paper, we focus on characterizing linearizability for data-differentiated executions, rather than arbitrary ones. The proof of this lemma can be found in Appendix B.

Lemma 2. *A data-independent implementation $\mathcal{I}$ is linearizable with respect to a data-independent specification S, if and only if $\mathcal{I}_{\neq}$ is linearizable with respect to $S_{\neq}$.*

5 Step-by-Step Linearizability of Extended Priority Queues

In this section we shows that, with the help of a property called step-by-step linearizability, we can partition the concurrent executions which are not linearizable with respect to *EPQ* into a finite number of classes. Intuitively, each such class represents a set of sequential execution that violate one rule of extended priority queue. Here step-by-step linearizability enables us to build a linearization for an execution e incrementally, using linearizations of projections of e. Our step-by-step linearizability is inspired by the step-by-step linearizabilility of queue and stacks in [1]. The proof of lemmas in this section can be found in Appendix C.

The projection $e|\mathcal{D}$ of a sequential execution e into a subset $\mathcal{D} \subseteq \mathbb{D}$ of items is obtained from e by erasing all method events with a data value not in $\mathcal{D}$. The set of projections of e is denoted *proj*(e). When refer to *proj*(e), we implicitly assume that each *rm*(*empty*) in e has a ghost argument that is unique. We write $e \backslash x$ for the projection $e|_{\mathbb{D} \backslash \{x\}}$. This extends naturally to histories and concurrent executions.

A set S of sequential executions is closed under projection, if for all $\mathcal{D} \subseteq \mathbb{D}$ and $e \in S$, we have $e|_{\mathcal{D}} \in S$. The following lemma states that *EPQ* is closed under projection, since the predicates used in rules of extended priority queue are "closed under projection".

Lemma 3. *EPQ is closed under projection.*

A sequential execution e matches a rule R of extended priority queue, if $e = \mathit{Expr}(l_1, \ldots, l_k, a, b)$, and $\mathit{Guard}(l_1, \ldots, l_k, a, b)$ holds. Here *Guard* and *Expr* are of rule R, and we call a (if exists) the witness of e. We denote by *MS*(R) the set of sequential executions which match R. Note that sequences in *MS*(R) only respect rule R and may be not in *EPQ*. e is linearizable with respect to *MS*(R) with witness x, if e is linearizable with respect to $u \in \mathit{MS}(R)$ and x is the witness of u.

Example 5. Assume that $p_1 <_{\mathbb{P}} p_2$, we can see that $e = \mathit{rm}(b) \cdot \mathit{put}(b, p_1) \cdot \mathit{put}(a, p_2) \cdot \mathit{rm}(a)$ is in $\mathit{MS}(\mathit{EPQ}_1^{>})$, but it is obvious that $e \notin \mathit{EPQ}$.

The following lemma states that for data-differentiated sequential execution, checking inclusion into *EPQ* is equivalent to checking inclusion into *MS*(R) for everyone of its projections.

Lemma 4. *Given a data-differentiated sequential execution e, $e \in$ EPQ, if and only if, $\forall e' \in$ proj(e) and $\forall R \in$ last(e'), we have $e' \in$ MS(R).*

Lemma 4 simplifies checking inclusion into *EPQ*, since checking *MS*(R) only concerns information of one rule, while checking *EPQ* need to consider every method events. We want a similar lemma for checking linearizability with respect to *EPQ*. To enable such equivalent characterization, we introduce the notion of step-by-step linearizability for extended priority queue. The projection $e|O$ of a concurrent execution e into a set O of operations is obtained from e by erasing all call and return actions of non-O operations. We write $e \setminus o$ for the projection $e|_{O_h \backslash \{o\}}$, where O_h is the set of operations of e. This extends naturally to histories. Similarly we can define projection into method events.

Definition 8. *A set S of sequential executions of extended priority queue is step-by-step linearizable, if for any data-differentiated execution e,*

- *if e is linearizable w.r.t. $MS(R)$ ($R \in \{EPQ_1^{>}, EPQ_1^{=}, EPQ_2^{>}, EPQ_2^{=}\}$) with witness x, we have: $e \setminus x \sqsubseteq EPQ \Rightarrow e \sqsubseteq EPQ$.*
- *if e is linearizable w.r.t. $MS(EPQ_3)$ and o is a $rm(empty)$ event, we have: $e \setminus o \sqsubseteq EPQ \Rightarrow e \sqsubseteq EPQ$.*

Given a data-differentiated execution and its history, we can abuse notation and mix labels and method events with operations themselves, since items are unique in a data-differentiated execution. For instance, we will reference an operation labeled by $put(p, a)$ as $put(p, a)$. The following lemma states that *EPQ* is step-by-step linearizability.

Lemma 5. *EPQ is step-by-step linearizability.*

Let us briefly explain the idea of proving step-by-step linearizability of EPQ_1 with an example, while the other two rules is much simpler to deal with. Given a data-differentiated concurrent execution $e \sqsubseteq l = u \cdot put(x, pri_x) \cdot v \cdot rm(x) \cdot w \in MS(EPQ_1)$ with witness x and assume that $e \setminus x \sqsubseteq l' \in EPQ$, we explicitly construct a sequence $l'' = l''_1 \cdot put(x, pri_x) \cdot l''_2 \cdot rm(x) \cdot l''_3$ and prove that $e \sqsubseteq l'' \in EPQ$. e is shown in Fig. 1 and we explicitly draw the linearization points according to l'. Here $pri_x = p_2$, and assume that $p_1 <_{\mathbb{P}} p_2$ and $p_1 <_{\mathbb{P}} p_3$. Here $u = \epsilon$, $v = put(z_1, p_3) \cdot put(x_2, p_2) \cdot rm(y_1) \cdot put(y_1, p_1) \cdot rm(z_2) \cdot rm(x_2)$, and $w = rm(z_2) \cdot put(x_3, p_2) \cdot rm(z_1)$. We use $o - w$ to emphasize that o is in w.

The construction of l'' is as follows. Our construction does not rely how to choose linearization points of l' in e.

- At first glance, we can construct l''_1, l''_2 and l''_3 as the projection of l' into operations of u, v and w, respectively. However, this is incorrect. To explain this, let pri_x-comparable operations (resp., pri_x-incomparable operations) be the operations with items whose priority is comparable with pri_x (resp., incomparable with pri_x). According to EPQ_1, there is no restriction to pri_x-incomparable operations operations in $u \cdot v$, and thus, there is no guarantee that the projection of l' into pri_x-incomparable operations operations in $u \cdot v$ being correct. In this example, we can see that such projection is $put(z_1, p_3) \cdot rm(z_2)$ and is incorrect.
- Let us construct set U', V' and W', such that l''_1, l''_2 and l''_3 are projection of l' into U', V' and W', respectively. The construction contains two steps:
- The first step is to define W'. The pri_x-comparable operations in W' is same as that in W. To obtain pri_x-incomparable operations in W', we try to find an operation o which either happens before some pri_x-comparable operations in W, or with linearization points after $rm(x)$. In this example, o is $put(x_3, p_2)$ (emphasized by adding vertical dashed line). Then, we put o and all pri_x-incomparable operations in l' whose linearization points are after o into W'. In this example, W' contains $put(x_3, p_2)$, $rm(z_1)$ and $rm(z_2)$. We use boxes to emphasize they are put into W'.
- The second step is to define U' and V'. U' contains the following two kinds of operations: (1) Operations whose linearization points are before $ret(put, x)$, and (2) other *put* operations with priority pri_x. V' contains the remanning operations. In this example, U' contains $put(z_1, p_3)$ and $put(x_2, p_2)$.

- In this example, $l''_1 = put(z_1, p_3) \cdot put(x_2, p_2)$, $l''_2 = put(z_2, p_3) \cdot rm(x_2) \cdot put(y_1, p_1) \cdot rm(y_1)$, and $l''_3 = put(x_3, p_2) \cdot rm(z_1) \cdot rm(z_2)$. In Fig. 2, we add linarization points according to l'', and we can see that l'' holds as required.

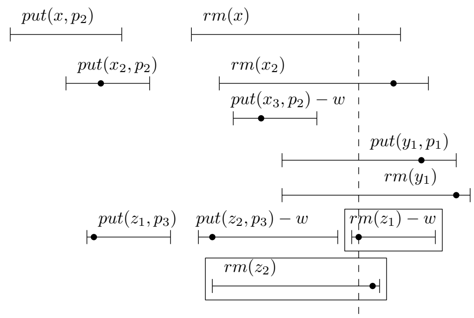

Fig. 1. Concurrent execution e

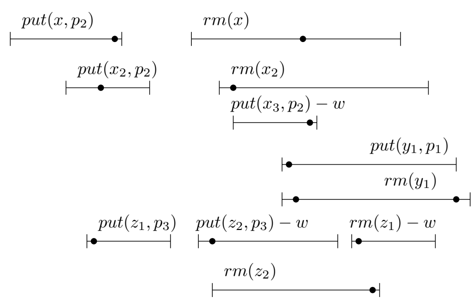

Fig. 2. Concurrent execution e with linearization points according to l''

The following lemma states that for data-differentiated execution, checking linearizability with respect to *EPQ* is equivalent to checking linearizability with respect to $MS(R)$ for everyone of its projections. Roughly speaking, step-by-step linearizability of extended priority queue play a important rule for proof of the *if* direction of Lemma 6: It guide us how to build a linearization of a whole execution by increasingly construct linearization of sub-execution from ϵ execution.

Lemma 6. *Given a data-differentiated execution e, $e \sqsubseteq$ EPQ, if and only if, $\forall e' \in$ proj(e) and $\forall R \in$ last(e'), we have $e' \sqsubseteq$ MS(R).*

6 Reducing Linearizability of Extended Priority Queues into State Reachability

A rule R of extended priority queue is co-regular, if checking linearizability with respect to $MS(R)$ of a data-independent implementation $\mathcal{I}$ can be reduced to checking the emptiness of intersection between $\mathcal{I}$ and a set of witness automata. In this section, we propose the definition of witness automata and co-regular. Then we prove that all five rules of extended priority queues are co-regular, and roughly introduce the proof idea. With the help of step-by-step linearizability and co-regular, we finally reduce the linearizability problem of *EPQ* into emptiness problem of intersection with automata.

6.1 Definition of Co-Regular

A witness automaton is a finite automaton with alphabet $\{cal(put, d, pred), ret(put, d), cal(rm, d), ret(rm, d) | d \in \mathbb{D} \cup \{empty\}, pred \in predWA\}$. Here *predWA* is the set of predicate of priorities, and it contains

- A predicate variable p, which accepts some specific priority,
- A predicate les_p, which accepts all the priorities that are smaller than the value of p according to $<_{\mathbb{P}}$,
- A predicate *anyPri*, which accepts any priority.

Given a execution $e = \alpha_1 \cdots \alpha_k$ of extended priority queue and a witness automaton $\mathcal{A}$, we say that e is accepted by $\mathcal{A}$, if

- There exist transitions $q_0 \xrightarrow{\beta_1} q_1 \ldots \xrightarrow{\beta_k} q_k$ of $\mathcal{A}$, such that q_0 is one of initial state of $\mathcal{A}$, and q_k is one of accept state of $\mathcal{A}$. We choose value of p as some value $d_p \in \mathbb{D}$.
- For each i, if $\alpha_i = cal(put, a, q)$, then either (1) $\beta_i = cal(put, a, p)$ and $q = d_p$, or (2) $\beta_i = cal(put, a, les_p)$ and $q <_{\mathbb{P}} d_p$, or (3) $beta_i = cal(put, a, anyPri)$.
- For each i, if $\alpha_i = ret(put, a), cal(rm, a), ret(rm, a), cal(rm, empty)$ or $ret(rm, empty)$, then $\beta_i = \alpha_i$.

Note that witness automata does not read operations, since the domains of operations is infinite.

Let us introduce the notion of co-regular:

Definition 9. *A rule R of extended priority queue is co-regular, if there are a finite set $Auts_R$ of witness automata such that, for each data-independence implementation $\mathcal{I}$, we have that*

$$Auts_R \cap \mathcal{I} \neq \emptyset \Leftrightarrow \exists e \in \mathcal{I}_{\neq}, e' \in proj(e), R \in last(e') \wedge e' \, does \, not \, linearizable \, w.r.t. \, MS(R)$$

We say that EPQ is co-regular, if each of its rule is co-regular.

Before we go to investigate co-regular of each rules, we use the results in [1] to simplify our work. [1] states that checking linearizability w.r.t queue can be reduced into checking emptiness of intersection between $\mathcal{I}$ and a set of automata. Given a data-differentiated execution e, let $e|_i$ be an execution generated from e by erasing call and return actions of items that does not use priority i (does not influence *rm*$(empty)$). We call a extended priority queue execution with only one priority a single-priority execution. Let *transToQueue*(e) be an execution generated from e by transforming *put* and *rm* into *enq* and *deq*, respectively, and then discarding priorities. We can see that for each $e \in$ *EPQ* and each priority i, *transToQueue*$(e|_i)$ satisfy FIFO (first in first out) property.

Given an execution of queue, we say that it is differentiated [3], if each item is enqueued at most once. [1] states that, given a differentiated queue execution e without *deq*$(empty)$, e is not linearizable with respect to queue, if one of the following cases holds for some a, b: (1) $deq(b) <_{hb} enq(b)$, (2) there are no $enq(b)$ and at least one $deq(b)$, (3) there are one $enq(b)$ and more than one $deq(b)$, and (4) $enq(a) <_{hb} enq(b)$, and $deq(b) <_{hb} deq(a)$, or $deq(a)$ does not exists. For each such case, we can construct a witness automata for extended priority queue. For example, for the first case, we generate witness automata $\mathcal{A}^1_{SinPri}$ in Fig. 3, here $c_1 = cal(put, a, anyPri), ret(put, a), cal(rm, a), ret(rm, a), cal(rm, b), cal(rm, empty), ret(rm, empty)$, $c_2 = c_1 + ret(rm, b)$, $c_3 = c_2 + ret(put, b)$.

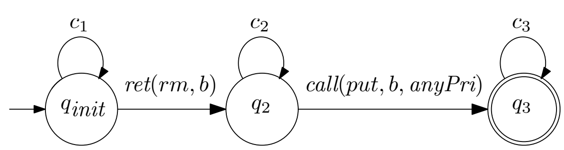

Fig. 3. Automaton $\mathcal{A}^1_{SinPri}$

In Appendix D.1, we construct a set $Auts_{sinPri}$ of witness automata ($\mathcal{A}^1_{SinPri}$ is in it), and shows that they are enough to ensure that for each data-differentiated executions, each of its single-priority projection without *rm*$(empty)$ to have "FIFO" property, as shown by the following lemma.

Lemma 7. *Given a data-independent implementations $\mathcal{I}$ of extended priority queue, $\mathcal{I} \cap Auts_{sinPri} \neq \emptyset$, if and only if there exists $e \in \mathcal{I}_{\neq}$, $e' \in proj(e)$, such that e' is single-priority without rm$(empty)$, and transToQueue(e') does not linearizable to queue.*

According to Lemma 7, from now on, it is safe to assume that, for each data-differentiated execution without *rm*$(empty)$, any of its single-priority projection has "FIFO" property. For example, *rm*(a) never happens before *put*$(a, \_)$ for each a.

6.2 Co-Regular of $EPQ^{>}_1$

In this subsection, we introduce the idea for proving co-regular of $PQ^{>}_1$. The proof of this subsection can be found in Appendix D.2. The notion of left-right constraint used for extended priority queue is inspired by left-right constraint of queue [1].

Given a data-differentiated execution e, we say that e is a *pri*-execution, if *pri* is the maximal priority of e, and *pri* is larger than or equal to the priority of all other items of e. Given a data-differentiated execution e and assume that *pri* is one of the maximal priority of e, let *pri-Exec*(e) be an execution obtained from e by erasing all operations of items whose priority are incomparable with *pri*. The following lemma states that when checking co-regular of $EPQ_1^{>}$, $EPQ_1^{=}$, $EPQ_2^{>}$ and $EPQ_2^{=}$, we need only consider *pri*-executions.

Lemma 8. *Given a data-differentiated execution* e *and* $R = EPQ_1^{>}$ *(resp.,* $R = EPQ_1^{=}$*,* $R = EPQ_2^{>}$*, or* $R = EPQ_2^{=}$*).* $e \sqsubseteq MS(R)$ *with witness* x*, if and only if pri-Exec*$(e) \sqsubseteq MS(R)$ *with witness* x*, where pri is the priority of* x*.*

Lemma 8 simplifies our proof by make us safely ignore all the items that has priorities incomparable with *pri*. When construct witness automata, priorities which are incomparable with *pri* can be safely represented by *anyPri*. Given a *pri*-execution e', we can see that *last*(e') contains only one rules.

Given a data-differentiated _-execution e such that *last*$(e) = EPQ_1^{>}$, although Lemma 7 ensures that each single-priority projection of e satisfy the FIFO property, this is still not enough for ensuring that $e \sqsubseteq MS(EPQ_1^{>})$. Since it is possible that e does not linearizable w.r.t $MS(EPQ_1^{>})$ because of interaction between actions of multiple priorities.

We give an example of such execution e in Fig. 4. We call the time interval from $ret(put, x)$ to $cal(rm, x)$, or from $ret(put, x)$ when $cal(rm, x)$ does not exist, the interval of item x. In Fig. 4, we draw the interval of each item by dashed line. Here we assume that $p_1 <_{\mathbb{P}} p_4$, $p_2 <_{\mathbb{P}} p_4$ and $p_3 <_{\mathbb{P}} p_4$. The reason of why e does not linearizable w.r.t $MS(PQ_1^{>})$ is that, each time point from $cal(rm, b)$ to $ret(rm, b)$ is in interval of some item with smaller priority.

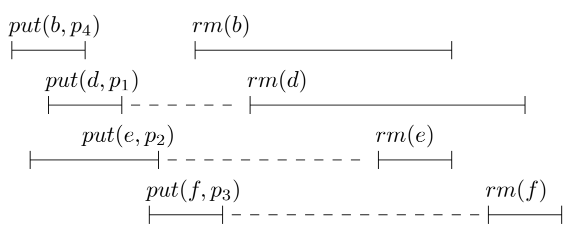

Fig. 4. An execution that does not linearizable w.r.t $MS(EPQ_1^{>})$

In this subsection, intuitively we will prove that, as long as we can get rid of the case in Fig. 4, we can ensure $last(e) = EPQ_1^{>} \Rightarrow e \sqsubseteq MS(EPQ_1^{>})$.

Let us introduce the notion of left-right constraint, which is a graph, and the existence of cycle though item with maximal priority in it correspond to the existence of the case in Fig. 4.

Definition 10. *Given a data-differentiated* pri_x*-execution* e *without* $rm(empty)$*. Let* $put(x, pri_x)$ *and* $rm(x)$ *be the only method events of* e *with priority* pri_x*. The left-right constraint of* $put(x, pri_x)$ *and* $rm(x)$ *is the graph* G *where:*

- *the nodes are the items of* e*, to which we add a node,*
- *there is an edge from item* d_1 *to* x*, if* $put(d_1, \_)$ *happens before* $put(x, pri_x)$ *or* $rm(x)$*,*
- *there is an edge from* x *to item* d_1*, if* $rm(x)$ *happens before* $rm(d_1)$ *or* $rm(d_1)$ *does not exists in* h*,*
- *there is an edge from item* d_1 *to item* d_2*, if* $put(d_1, \_)$ *happens before* $rm(d_2, \_)$*.*

When there is a cycle $d_1 \rightarrow \ldots \rightarrow d_m \rightarrow x \rightarrow d_1$ through item with maximal priority (for example, x) in G, we say that x is covered by $d_1, \ldots, d_m$. To state the effectiveness of left-right constraint, take the execution in Fig. 4 as an example, we can see that b is covered by f, e, d. We need to prove that getting rid of cycle though item with maximal priority in left-right constraint is enough for ensure linearizable w.r.t $MS(EPQ_1^{>})$, as stated by the following lemma:

Lemma 9. *Given a data-differentiated* pri_x*-execution* e *with* $last(e) = EPQ_1^{>}$*. Let* $put(x, pri_x)$ *and* $rm(x)$ *be method events of* e *with maximal priority. Let G be the graph representing the left-right constraint of* $put(x)$ *and* $rm(x)$*.* $e \sqsubseteq MS(EPQ_1^{>})$*, if and only if G has no cycle going through* x*.*

Proof. (Sketch)

The *only if* direction can be easily proved by contradiction and is omitted here. To prove the *if* direction, we need to explicitly construct the linearization of e, or we can say, we need construct the u, v and w in $EPQ_1^{>}$. Let u to be the sequence of all operations that happens before $put(x)$. The difficulties is how to generate proper v.

To generate v, we introduce $UVSet(e, x)$, which intuitively contains all pairs of method events that should be putted before $rm(x)$. Let $UVSet_1(e, x) = \{o |$ either $o <_{hb} put(x)$ or $rm(x)$, or $\exists o'$ with the same item of o, such that $o' <_{hb} put(x)$ or $rm(x)\}$. For each $i \geq 1$, let $LMSet_{i+1}(e, x) = \{o | \ o \notin UVSet_k(e, x)$ for each $k \leq i$, and either o happens before some operation $o' \in UVSet_i(e, x)$, or $\exists o''$ with the same item of o and o'' happens before some operation $o' \in UVSet_i(e, x)\}$. Let $UVSet(e, x) = UVSet_1(e, x) \cup UVSet_2(e, x) \cup \ldots$. For example, assume that $p_1 <_{\mathbb{P}} p_2$, then in Fig. 5, $UVSet_1(e, x) = \{put(a, p_1), rm(a)\}$, $USSet_2(e, x) = \{put(b, p_1), rm(b)\}$ and $UVSet_3(e, x) = \{put(c, p_1), rm(c)\}$. Similarly, we can generate execution e' such that, for each i, $UVSet(e', x) \neq \emptyset$.

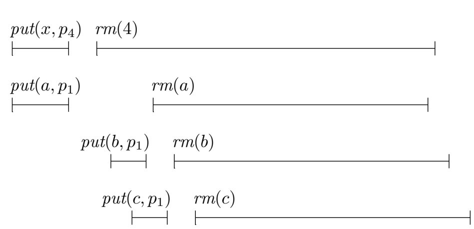

Fig. 5. An example for $UVSet(e, x)$

We let the v to be the sequence of method events that are in $\mathit{UVSet}(e, x)$ and not in u. We let w to be the sequence of remaining method events. Then we can prove that such u, v and w holds as required. $\square$

According to Lemma 9, the existence of "An item covered by items of smaller priority" is enough for checking violation of linearizability w.r.t $\mathit{MS}(\mathit{EPQ}_1^{>})$. Assume x is covered by $d_1, \ldots, d_m$. Then we can safely rename x into b, rename $d_1, \ldots, d_m$ into a and rename all other item into c by data-independence. Such execution can be recognized by witness automata, since between the first $\mathit{ret}(\mathit{put}, a, \_)$ and the last $\mathit{cal}(\mathit{rm}, a)$ (if exists), $\mathit{ret}(\mathit{put}, a, \_)$ and $\mathit{cal}(\mathit{rm}, a)$ occurs in pair and there is no need for count.

There are four possible enumeration of call and return actions of $\mathit{put}(b)$ and $\mathit{rm}(b)$. For each of them, we generate a witness automaton. For example, for the case when $e|_b = \mathit{cal}(\mathit{put}, b, p) \cdot \mathit{ret}(\mathit{put}) \cdot \mathit{cal}(\mathit{rm}) \cdot \mathit{ret}(\mathit{rm}, b)$, we generate witness automaton $\mathcal{A}^1_{\mathit{l\text{-}lar}}$, as shown in Fig. 14. Here $c_1 = c + \mathit{ret}(\mathit{rm}, a)$, $c_2 = c + \mathit{cal}(\mathit{put}, a, \mathit{les}_p)$, $c_3 = c_2 + \mathit{ret}(\mathit{rm}, a)$, where $c = \mathit{cal}(\mathit{put}, d, \mathit{anyPri}), \mathit{ret}(\mathit{put}, d), \mathit{cal}(\mathit{rm}, d), \mathit{ret}(\mathit{rm}, d), \mathit{cal}(\mathit{rm}, \mathit{empty}), \mathit{ret}(\mathit{rm}, \mathit{empty})$.

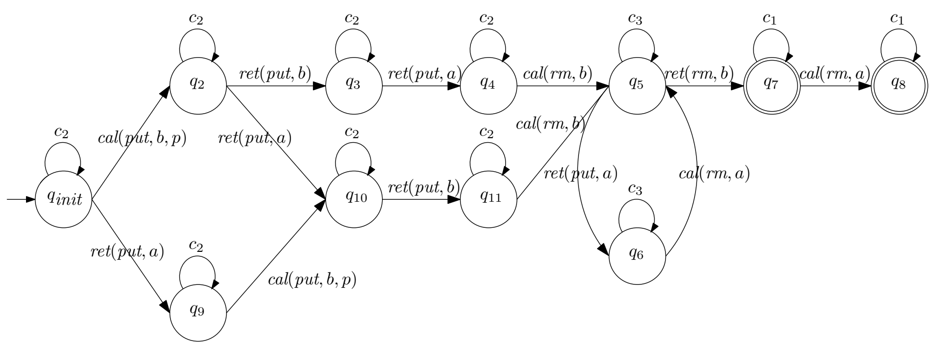

Fig. 6. Automaton $\mathcal{A}^1_{\mathit{l\text{-}lar}}$

Given the execution e in Fig. 4, let e' be generated from e by renaming d, e, f into a. Then we can see that e' is accepted by $\mathcal{A}^1_{\mathit{l\text{-}lar}}$ with a path via $q_{\mathit{init}}, q_2, q_3, q_4, q_5, q_7, q_8$.

In Appendix D.2, we construct a set $\mathit{Auts}_{\mathit{1\text{-}lar}}$ of witness automata ($\mathcal{A}^1_{\mathit{l\text{-}lar}}$ is in it), and use $\mathit{Auts}_{\mathit{1\text{-}lar}}$ to prove that $\mathit{EPQ}_1^{>}$ is co-regular, as stated by the following lemma.

Lemma 10. *$\mathit{EPQ}_1^{>}$ is co-regular.*

6.3 Co-Regular of $\mathit{EPQ}_1^{=}$

In this subsection, we introduce the idea for proving co-regular of $\mathit{EPQ}_1^{=}$. The proof of this subsection can be found in Appendix D.3.

Given a data-differentiated $\_$-execution e such that $\mathit{last}(e) = \mathit{EPQ}_1^{=}$. Lemma 7 ensures that each single-priority projection of e satisfy the FIFO property, and with the help of Lemma 10 and $\mathit{Auts}_{\mathit{1\text{-}lar}}$, we can ensure that each item in e can not be covered by items with smaller priority. However, these are sill not enough for ensure that

$e \sqsubseteq MS(EPQ_1^{=})$. This is because that given items a and b with maximal priority, it is possible that all the possible linearization point of *rm*(b) may be disabled by *rm*(b).

We give an example of such execution e in Fig. 7. The execution e of Fig. 7 is not linearizable, even if $h|_{p_1}$ and $h|_{p_4}$ are both linearizable, and either a or b is covered by items with smaller priority. To explain this, me need to observe the following three points:

- Since *put*(a, p_4) happens before *put*(b, p_4), we know that a is "putted earlier" than b, and therefore, the linearization points of *rm*(a) should before the linearization point of *rm*(b).
- The time intervals satisfy the following conditions (we identify them with dotted lines in Fig. 7) are possible position to locate the linearization point of *rm*(b) according to Lemma 9 (Here we temporarily forget the existence of a): (1) between *cal*(rm, b) and *ret*(rm, b), (2) does not in interval of any item with smaller priority.
- To satisfy requirement of the second condition, the linearization points of *rm*(b) should be in the time interval of dotted line. However, all time point in dotted line is before *cal*(rm, a) and is thus disabled by *cal*(rm, a).

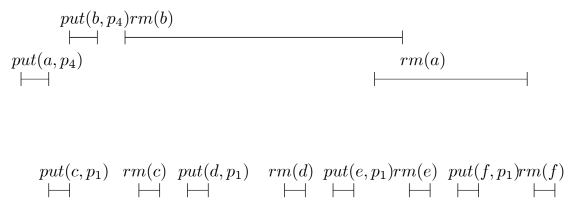

Fig. 7. An execution that does not linearizable w.r.t $MS(EPQ_1^{=})$

In this subsection, intuitively we will prove that, as long as we can get rid of the case in Fig. 7, we can ensure *last*$(e) = EPQ_1^{=} \Rightarrow e \sqsubseteq MS(EPQ_1^{=})$.

Let us introduce $<_{pb}$ (represents put-before) order to formally state that "an item is putted before another item". Given a data-differentiated _-execution e and two items a, b with maximal priority in e, we say that $a <_{pb} b$, if one of the following cases holds:

- *put*$(a, \_)$ happens before *put*$(b, \_)$ in e,
- *rm*(a) happens before *rm*(b) in e,
- *rm*(a) happens before *put*$(b, \_)$ in e,

Sometimes we use $a <_{pb}^{A} b$, $a <_{pb}^{B} b$ and $a <_{pb}^{C} b$ to explicitly distinguish above three cases. Let $<_{pb}^{*}$ be the transitive closure of $<_{pb}$. Intuitively, $a <_{pb}^{*} b$ means that we can infer that a should be inserted earlier than b with the help of possibly other items.

Let us introduce gap-point to formally define the time point in dotted line of Fig. 7.

Definition 11. *Given a data-differentiated pri$_x$-execution e and two method events put(x, pri_x), rm(x) of e. We say that a time-point o is a gap-point of x, if*

- *o is after $call(put, x, pri_x)$ and $call(rm, x)$, and is before $ret(rm, x)$.*
- *o is not in interval of any items with smaller priority.*

Going back to Fig. 7, we can effectively characterize it using $<_{pb}$ order and gap-point as follows: $a <_{pb} b$, while the right-most gap-point of b is before $cal(rm, a)$. We need to prove that getting rid of such case is enough fo ensure linearizable w.r.t $MS(PQ_1^{=})$, as stated by the following lemma. Let $Items(e, p)$ be the set of items with priority p in execution e.

Lemma 11. *Given a data-differentiated pri-execution e with $last(e) = EPQ_1^{=}$. e does not linearizable w.r.t $MS(EPQ_1^{=})$, if and only if there exists x and y with maximal priority pri in e, such that $y <_{pb}^{*} x$ in e, and the rightmost gap-point of x is before $cal(put, y, pri)$ or $cal(rm, y)$ in e.*

Proof. (Sketch)

We have already intuitively explain the proof of the *if* direction.

To prove the *only if* direction, we prove its contrapositive. Assume we already know that for each x and y has maximal priority in e, if $y <_{pb}^{*} x$, then the rightmost gap-point of x is after $cal(put, y, pri)$ and $cal(rm, x)$. We need to prove that $e \sqsubseteq MS(EPQ_1^{=})$. Or we can say, we need to explicitly construct linearization of e.

We introduce another lemma (Lemma 27), which states that: If $\exists g_1 \in Items(e, pri)$, such that $\forall g_2 \in Items(e, pri)$, (1) g_1 does not $<_{pb}$ to g_2, and (2) the right-most gap-point of g_1 is after $cal(put, g_2, pri)$ and $cal(rm, g_2)$. Then $e \sqsubseteq MS(EPQ_1^{=})$. The proof of this lemma also tell us how to construct linearization in such case.

Since every single-priority projection has FIFO property, let l_{pri} be the linearization of projection of e into actions of priority pri. Then our proof proceeds as follows:

- We start with a_1, the last inserted item of l_{pri}.
- Step 1: check if a_1 satisfy the conditions of 27.
  It is obvious that a_1 satisfy the first condition. If the second condition is also satisfied, by Lemma 27, we can obtain that $e \sqsubseteq MS(EPQ_1^{=})$.
- Otherwise, $\exists a_2 \in Items(e, pri)$, such that the rightmost gap-point of a_1 is before $cal(put, a_2, pri)$ or $cal(rm, a_2)$ in e. We can see that each gap-point of a_2 is after the rightmost gap-point of a_1. By assumption, we know that a_2 does not $<_{pb}$ to a_1.
  - If $\forall b \in Items(e, pri)$, a_2 does not $<_{pb}$ to b. Then we go to step 1 and treat a_2 similarly as a_1.
  - Otherwise, there exists a_3 with priority *pri* such that $a_2 <_{pb}^{*} a_3$.
    Since l_{pri} has FIFO property, it is easy to see that there is no cycle in $<_{pb}$ order. It is safe to assume that a_3 is maximal in the sense of $<_{pb}^{*}$. Or we can say, there does not exists a_4, such that $a_3 <_{pb}^{*} a_4$.
    By assumption,we know that the rightmost gap-point of a_3 is after $cal(put, a_2, pri)$ and $cal(rm, a_2)$. Therefore, we can see that the rightmost gap-point of a_3 is after the rightmost gap-point of a_1. Then we go to step 1 and treat a_3 similarly as a_1.

Let a^i be the a_1 in the *i-th* loop of our proof. It is not hard to see that, given $i < j$, the rightmost gap-point of a^j is after the rightmost gap-point of a^i. Therefore, the loop finally stop at some a^f. a^f satisfies the check of Step 1. By Lemma 27, this implies that $e \sqsubseteq MS(EPQ_1^{=})$. This completes the proof of *if* direction. □

The result of Lemma 11 is not quite suitable for verificaion with automata. The reason is that we need to ensure $y <^*_{pb} x$, while according to definition of $<^*_{pb}$, there may be arbitrary intermediate items $a_1, \ldots, a_m$, such that $x <_{pb} a_1 <_{pb} \ldots <_{pb} a_m <_{pb} y$. It is hard to store unbounded a_i by automata. Fortunately, we find that the number of intermediate items a_i is in fact bounded, as stated by the following lemma.

Lemma 12. *Given a data-differentiated execution h. Assume that $a <_{pb} a_1 <_{pb} \ldots <_{pb} a_m <_{pb} b$, then one of the following cases holds:*

- *$a <^A_{pb} b$, $a <^B_{pb} b$ or $a <^C_{pb} b$,*
- *$a <^A_{pb} a_i <^B_{pb} b$, or $a <^B_{pb} a_i <^A_{pb} b$, for some i.*

With Lemma 12, now we can use witness automata to detect $a <^*_{pb} b$ by enumerating all possible enumerations of a, b and an intermediate item a_1, and then check if it satisfy one cases of 12.

The number of potential enumerations can be further reduced. The reason is that Lemma 11 requires both $<^*_{pb}$ and gap-point, while some combination of $<^*_{pb}$ and gap-points are conflict. For example, if $a <^B_{pb} b$, and the rightmost gap-point of b is before $cal(put, a, \_)$ or $cal(rm, a)$, then since (1) $rm(a) <_{hb} rm(b)$, (2) each gap-point of b is after $cal(rm, b)$ (also $ret(rm, a)$), we can see that the right-most gap-point of b is before $cal(put, a, \_)$ and after $ret(rm, a)$, which implies $rm(a) <_{hb} put(a, \_)$ and is impossible. Therefore, we finally reduce the number of potential enumeration into only five, as shown by the following lemma:

Lemma 13. *Given a data-differentiated pri-execution e with $last(e) = EPQ^{=}_1$. Let a and b be items with maximal priority pri. Assume that $a <^*_{pb} b$, and the rightmost gap-point of b is before $cal(put, a, pri)$ or $cal(rm, a)$. Then, there are five possible enumeration of method events of a, b, a_1 (if exists), where a_1 is the possible intermediate items for obtain $a <^*_{pb} b$.*

The five enumerations is shown in Fig. 8, where we use o to explicitly denote the right-most gap-point of b. In this figure, e_1 and e_2 comes from $a <^A_{pb} b$, e_3, e_4, e_5 comes from $a <^A_{pb} a_1 <^B_{pb} b$, while other reasons of $a <^*_{pb} b$ turns out to be either redundant or impossible.

For each enumeration in Fig. 8, we can generate several automata to capture it. For example, to capture the first enumeration, we generate witness automaton $\mathcal{A}^1_{l\text{-}eq}$, as shown in Fig. 9. Here we rename the items that "covers the time interval from $cal(rm, a)$ to $ret(rm, b)$" (see Appendix D.3) into d, and rename the renaming items into e. In this figure, $c = cal(put, e, anyPri), ret(put, e), cal(rm, e), ret(rm, e), cal(rm, empty), ret(rm, empty)$, $c_1 = c + cal(put, d, les_p)$, $c_2 = c_1 + ret(put, b)$, $c_3 = c_2 + ret(rm, d)$, $c_4 = c + ret(put, b) + ret(rm, d)$.

In Appendix D.3, we construct a set $Auts_{1\text{-}eq}$ of witness automata ($\mathcal{A}^1_{l\text{-}eq}$ is in it), and use $Auts_{1\text{-}eq}$ to prove that $EPQ^{=}_1$ is co-regular, as stated by the following lemma.

Lemma 14. *$EPQ^{=}_1$ is co-regular.*

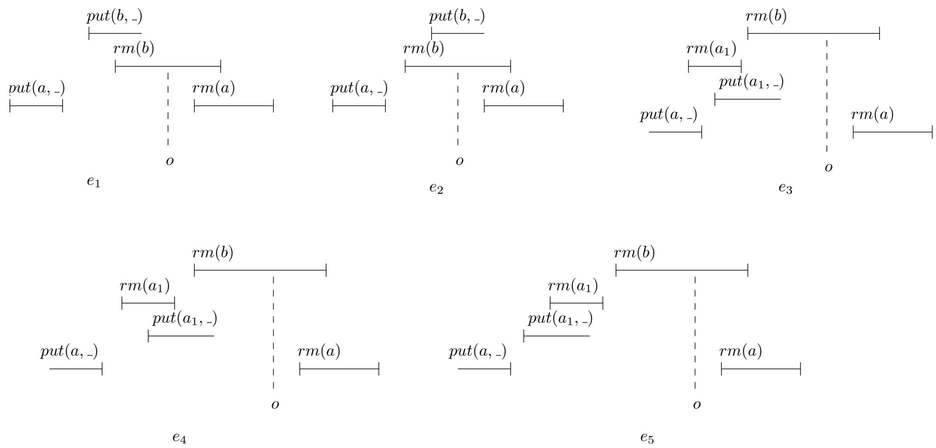

Fig. 8. Five possible enumerations

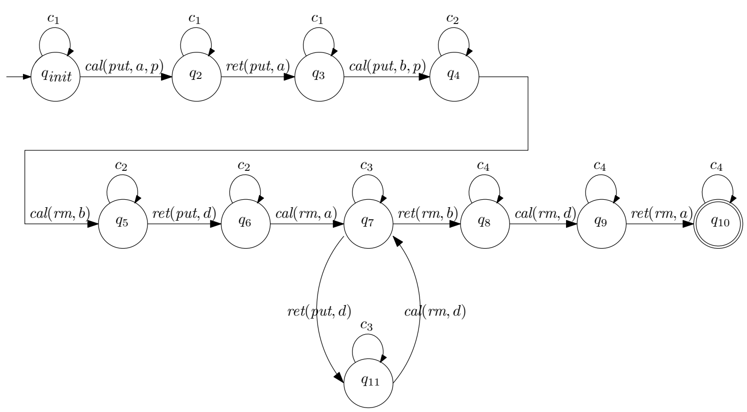

Fig. 9. Automaton $\mathcal{A}^1_{l\text{-}eq}$

6.4 Combine Step-by-Step linearizability and Co-Regular

We also prove that $\mathit{EPQ}_2^{>}$, $\mathit{EPQ}_2^{=}$ and EPQ_3 are co-regular. The case of $\mathit{EPQ}_2^{>}$ is quite simple, since whenever $\mathit{last}(e) = \mathit{EPQ}_2^{>}$, $e \sqsubseteq \mathit{MS}(\mathit{EPQ}_2^{>})$. For the case of $\mathit{EPQ}_2^{=}$, we find a necessary and sufficient condition for violating $\mathit{MS}(\mathit{EPQ}_2^{>})$, and shows that such condition violates the assumptions that every single-priority execution is FIFO and can be safely ignored. For the case of EPQ_3, it can be similarly proved as the rule for $\mathit{deq}(\mathit{empty})$ in [1]. We leave the detailed proof of co-regular of these three rules in Appendix D.4, Appendix D.5 and Appendix D.6, respectively.

Now we can see that *EPQ* is co-regular, which is a direct consequence of co-regular of each of its rules, and is stated below.

Lemma 15. *EPQ is co-regular.*

Since we have already prove that *EPQ* is step-by-step linearizability and co-regular, we can now reduce the verification of linearizability with respect to *EPQ* into a reachability problem, as illustrated by the following theorem. Here $\mathit{Auts}_{\mathit{EPQ}}$ is the set of witness automata of this section.

Theorem 1. *Given a data-independence implementation $\mathcal{I}$. $\mathcal{I} \sqsubseteq \mathit{EPQ}$, if and only if, $\mathcal{I} \cap \mathit{Auts}_{\mathit{EPQ}} = \emptyset$.*

References

1. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: On reducing linearizability to state reachability. In: Halldórsson, M.M. et al. (eds.) ICALP 2015, Part II, pp. 95–107. Springer (2015)
2. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Tractable Refinement Checking for Concurrent Objects. In: *POPL'15*. pp.651–662. (2015)
3. Wolper, P.: Expressing interesting properties of programs in propositional temporal logic. In *POPL'86*. pp.184–193. (1986)
4. Henzinger A. T., Sezgin A., Vafeiadis, V.: Aspect-Oriented Linearizability Proofs. In *CONCUR'13*. pp.242–256. (2013)
5. Abdulla, P. A., Haziza, F., Hol¨ak, L., Jonsson, B., Rezine, A.: An integrated specification and verification technique for highly concurrent data structures. In *TACAS'13*. pp.324–338. (2013)

A Proof in Section 3

A *labelled transition system* (LTS) is a tuple $\mathcal{A} = (Q, \Sigma, \rightarrow, q_0)$, where Q is a set of states, Σ is an alphabet of transition labels, $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation and q_0 is the initial state.

Let us model extended priority queue as an LTS $\mathit{LTS}_e = (Q, \Sigma, \rightarrow, q_0)$ as follows:

- Each state of Q is a function from $\mathbb{P}$ into a finite sequence over $\mathbb{D}$.
- The initial state q_0 is a function that maps each element in $\mathbb{P}$ into ϵ.
- $\Sigma = \{\mathit{put}(a,p), \mathit{rm}(a), \mathit{rm}(\mathit{empty}) | a \in \mathbb{D}, p \in \mathbb{P}\}$.
- The transition relation $\rightarrow$ is defined as follows:
  - $q_1 \xrightarrow{\mathit{put}(a,p)} q_2$, if q_1 maps p into some finite sequence l, and q_2 is the same as q_1, except for p, where it maps p into $a \cdot l$.
  - $q_1 \xrightarrow{\mathit{rm}(a)} q_2$, if q_1 maps p into $l \cdot a$ for some finite sequence l, and q_2 is the same as q_1, except for p, where it maps p into l. We also require that for each priority p' such that $p' <_{\mathbb{P}} p$, q_1 and q_2 map p' into ϵ.
  - $q_1 \xrightarrow{\mathit{rm}(\mathit{empty})} q_2$, if $q_1 = q_2$, and they maps each element in $\mathbb{P}$ into ϵ.

A path of an LTS is a finite transition sequence $q_0 \xrightarrow{\beta_1} q_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_k} q_k$ for $k \geq 0$, where q_0 is the initial state of the LTS. A trace of an LTS is a finite sequence $\beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k$, where $k \geq 0$ if there exists a path $q_0 \xrightarrow{\beta_1} q_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_k} q_k$ of the LTS. Let EPQ_s be the set of traces of LTS_e. The following lemma states that the language generated by our rules equals the set of traces of EPQ_s.

Lemma 1. $\mathit{EPQ} = \mathit{EPQ}_s$.

Proof. To prove that $\mathit{EPQ} \subseteq \mathit{EPQ}_s$, we prove that each sequence in EPQ is also in EPQ_s by induction:

- It is obvious that $\epsilon \in \mathit{EPQ}_s$.
- If $l_1 \in \mathit{EPQ}_s$ and $l_1 \xrightarrow{\mathit{EPQ}_1} l_2$. Then we need to prove that $l_2 \in \mathit{EPQ}_s$. Let $l_1 = u \cdot v \cdot w$, such that $\mathit{Guard}(u, v, w, \mathit{item}, \mathit{pri})$ of EPQ_1 holds, and $l_2 = u \cdot \mathit{put}(\mathit{itm}, \mathit{pri}) \cdot v \cdot \mathit{rm}(\mathit{itm})$.

  Assume that $u = \alpha_1 \cdot \ldots \cdot \alpha_i$, $v = \alpha_{i+1} \cdot \ldots \cdot \alpha_j$ and $w = \alpha_{j+1} \cdot \ldots \cdot \alpha_m$. Assume that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_j} q_j \xrightarrow{\alpha_{j+1}} q_{j+1} \ldots \xrightarrow{\alpha_m} q_m$ is the path of l_1 on EPQ_s. For each $i \leq k \leq j$, let q'_k be the same as q_k, except that q'_k maps *pri* into $\mathit{item} \cdot l_k$ and q_k maps *pri* into l_k for some finite sequence l_k.

  We already know that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i$, and it is obvious that $q_i \xrightarrow{\mathit{put}(\mathit{itm},\mathit{pri})} q'_i$. Since (1) all *put* with priority *pri* is in u, and (2) in $u \cdot v$, only items with priority either incomparable, or less, or equal than *pri* is removed, we can see that it is safe to add to each q_k ($1 \leq k \leq j$) with a newest *itm* with priority *pri*. Or we can say, $q'_i \xrightarrow{\alpha_{i+1}} q'_{i+1} \ldots \xrightarrow{\alpha_j} q'_j$ are transitions of LTS_e. Since $\mathit{matched\text{-}C}(u \cdot v)$ holds, we can see that q_j maps each priority that is smaller than *pri* into ϵ and maps *pri* into ϵ, and q'_j maps each priority that is smaller than *pri* into ϵ and maps *pri* into *itm*. Then, we can see that $q'_j \xrightarrow{\mathit{rm}(\mathit{itm})} q_j$. We already know that that $q_j \xrightarrow{\alpha_{j+1}} q_{j+1} \ldots \xrightarrow{\alpha_m} q_m$. Therefore, we can see that $l_2 = u \cdot \mathit{put}(\mathit{itm}, \mathit{pri}) \cdot v \cdot \mathit{rm}(\mathit{itm}) \in \mathit{EPQ}_s$.

- If $l_1 \in \mathit{EPQ}_s$ and $l_1 \xrightarrow{\mathit{EPQ}_2} l_2$. Then we need to prove that $l_2 \in \mathit{EPQ}_s$. Let $l_1 = u \cdot v$, such that $\mathit{Guard}(u, v, \mathit{item}, \mathit{pri})$ of EPQ_2 holds, and $l_2 = u \cdot \mathit{put}(\mathit{itm}, \mathit{pri}) \cdot v$.
  Assume that $u = \alpha_1 \cdot \ldots \cdot \alpha_i$ and $v = \alpha_{i+1} \cdot \ldots \cdot \alpha_m$. Assume that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_m} q_m$ is the path of l_1 on EPQ_s. For each $i \leq k \leq m$, let q'_k be the same as q_k, except that q'_k maps pri into $\mathit{item} \cdot l_k$ and q_k maps pri into l_k for some finite sequence l_k.
  We already know that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i$, and it is obvious that$q_i \xrightarrow{\mathit{put}(\mathit{itm},\mathit{pri})} q'_i$. Since (1) all *put* with priority *pri* is in u, (2) in $u \cdot v$, only items with priority either incomparable, or less, or equal than *pri* is removed, we can see that it is safe to add to each q_k $(1 \leq k \leq m)$ with a newest *itm* with priority *pri*. Or we can say, $q'_i \xrightarrow{\alpha_{i+1}} q'_{i+1} \ldots \xrightarrow{\alpha_m} q'_m$ are transitions of LTS_e. Therefore, we can see that $l_2 = u \cdot \mathit{put}(\mathit{itm}, \mathit{pri}) \cdot v \in \mathit{EPQ}_s$.
- If $l_1 \in \mathit{EPQ}_s$ and $l_1 \xrightarrow{\mathit{EPQ}_3} l_2$. Then we need to prove that $l_2 \in \mathit{EPQ}_s$. Let $l_1 = u \cdot v$, such that $\mathit{Guard}(u, v)$ of EPQ_3 holds, and $l_2 = u \cdot \mathit{rm}(\mathit{empty}) \cdot v$.
  Assume that $u = \alpha_1 \cdot \ldots \cdot \alpha_i$ and $v = \alpha_{i+1} \cdot \ldots \cdot \alpha_m$. Assume that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_m} q_m$ is the path of l_1 on EPQ_s.
  We already know that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i$. Since $\mathit{matched\text{-}All}(u)$ holds, we can see that q_i maps each element in $\mathbb{P}$ into ϵ, and then $q_i \xrightarrow{\mathit{rm}(\mathit{empty})} q_i$. We already know that $q_i \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_m} q_m$. Therefore, we can see that $l_2 = u \cdot \mathit{rm}(\mathit{empty}) \cdot v \in \mathit{EPQ}_s$.

To prove that $\mathit{EPQ}_s \subseteq \mathit{EPQ}$, we show that given $l_2 \in \mathit{EPQ}_s$ and $l_2 \neq \epsilon$, how to construct a sequence l_1, such that $l_1 \xrightarrow{R} l_2$ for some rule R, and $l_1 \in \mathit{EPQ}_s$. Based on this, we can decompose a sequence of EPQ_s into ϵ, while the reverse process is the reason of why this sequence is in *EPQ*. Note that from a l_2 we may construct more than one l_1, and this does not influence the correctness of our proof.

- If l_2 contains $\mathit{rm}(\mathit{empty})$: Assume that $l_2 = u \cdot \mathit{rm}(\mathit{empty}) \cdot v$. It is easy to see that $\mathit{matched\text{-}All}(u)$ holds. Let $l_1 = u \cdot v$. It is easy to see that $l_1 \xrightarrow{\mathit{EPQ}_3} l_2$, and l_1 satisfy the guard of EPQ_3.
  Assume that $u = \alpha_1 \cdot \ldots \cdot \alpha_i$ and $v = \alpha_{i+1} \cdot \ldots \cdot \alpha_m$. Since We already know that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\mathit{rm}(\mathit{empty})} q'_i \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_m} q_m$ is transitions of LTS_e. Since $\mathit{matched\text{-}All}(u)$ holds, we can see that $q_i = q'_i$, and they map each element in $\mathbb{P}$ into ϵ. Then we can see that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_m} q_m$ is transitions of LTS_e, and $l_1 \in \mathit{EPQ}_s$.
- If l_2 does not contain $\mathit{rm}(\mathit{empty})$, *pri* is one of maximal priority of l_2, and items in l_2 with priority *pri* are unmatched *put* and (possibly) matched *put*:
  Assume that $l_2 = u \cdot \mathit{put}(\mathit{itm}, \mathit{pri}) \cdot v$, such that all *put* with priority *pri* of $u \cdot v$ is in u. Let $l_1 = u \cdot v$. According to construction of LTS_e, we can see that l_1 satisfies the guard of EPQ_2, and $l_1 \xrightarrow{\mathit{EPQ}_2} l_2$.
  Assume that $u = \alpha_1 \cdot \ldots \cdot \alpha_i$ and $v = \alpha_{i+1} \cdot \ldots \cdot \alpha_m$. We already know that $\mathit{pa} = q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\mathit{put}(\mathit{itm},\mathit{pri})} q_{i+} \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_m} q_m$ are transitions of LTS_e. For each $i{+}1 \leq k \leq m$, let q'_k be the same as q_k, except that q_k maps

pri into some $itm \cdot l_k$ for some finite sequence l_k, and q'_k maps *pri* into l_k. Since (1) all *put* with priority *pri* of $u \cdot v$ is in u and (2) *pri* is one of maximal priority of l_2, it is safe to remove *itm* without influence other transitions of *pa*. Or we can say, $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\alpha_{i+1}} q'_{i+1} \ldots \xrightarrow{\alpha_m} q'_m$ are transitions of LTS_e. Therefore, $l_1 \in EPQ_s$.

- If l_2 does not contain *rm*(*empty*), *pri* is one of maximal priority of l_2, and items in l_2 with priority *pri* are matched *put*:
  Assume that $l_2 = u \cdot put(itm, pri) \cdot v \cdot rm(itm) \cdot w$, such that all *put* with priority *pri* of $u \cdot v \cdot w$ is in u. Let $l_1 = u \cdot v \cdot w$. According to construction of LTS_e, we can see that l_1 satisfies the guard of EPQ_1. We can also see that $l_1 \xrightarrow{EPQ_1} l_2$.
  Assume that $u = \alpha_1 \cdot \ldots \cdot \alpha_i$, $v = \alpha_{i+1} \cdot \ldots \cdot \alpha_j$ and $w = \alpha_{j+1} \cdot \ldots \cdot \alpha_m$. We already know that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{put(itm,pri)} q_{i+} \xrightarrow{\alpha_{i+1}} q_{i+1} \ldots \xrightarrow{\alpha_j} q_j \xrightarrow{rm(itm)} q_{j+} \xrightarrow{\alpha_{j+1}} q_{j+1} \ldots \xrightarrow{\alpha_m} q_m$. For each $i{+}1 \leq k \leq j$, let q'_k be the same as q_k, except that q_k maps *pri* into $itm \cdot l_k$ for some finite sequence l_k, and q'_k maps *pri* into l_k. Since (1) *pri* is one of maximal priority in l_2, (2) *itm* is the newest item with priority *pri* in l_2, and (3) *itm* is not removed until *rm*(*itm*), we know that whether we keep *itm* or remove it will not influence transitions from q_{i+1} to q_j. Then we can see that $q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_i} q_i \xrightarrow{\alpha_{i+1}} q'_{i+1} \ldots \xrightarrow{\alpha_j} q'_j \xrightarrow{\alpha_{j+1}} q_{j+1} \ldots \xrightarrow{\alpha_m} q_m$ are transitions of LTS_e. Therefore, $l_1 \in EPQ_s$.

This completes the proof of this lemma. □

B Proof in Section 4

Lemma 2. *A data-independent implementation $\mathcal{I}$ is linearizable with respect to a data-independent specification S, if and only if $\mathcal{I}_{\neq}$ is linearizable with respect to $S_{\neq}$.*

Proof. To prove the *only if* direction, given a data-differentiated execution $e \in \mathcal{I}_{\neq}$. By assumption, it is linearizable with respect to a sequential execution $l \in S$, and the bijection between the operations of e and the method events of l ensures that l is differentiated and belongs to $S_{\neq}$.

To prove the *if* direction, given an execution $e \in \mathcal{I}$. By data independence of $\mathcal{I}$, we know that there exists $e' \in \mathcal{I}_{\neq}$ and a renaming function r, such that $r(e') = e$. By assumption, e' is linearizable with respect to a sequential execution $l' \in S_{\neq}$. Let $l = r(l')$. By data independence of S it is easy to see that $l \in S$, and it is easy to see that $e \sqsubseteq l$ using the same bijection used for $e' \sqsubseteq l'$. □

C Proofs in Section 5

C.1 Proof of Lemma 3

Lemma 3. *EPQ is closed under projection.*

Proof. This is obvious, since for each conditions in the *Guard* part of the rules of priority queue, if a sequence of sequential executions satisfy it, then its sub-sequence also satisfy it. For example, if *noRE*(l) holds for some $l = u \cdot v \cdot w$ and let D_l be the set of items of l, then for each subset $D' \subseteq D_l$, it is obvious that *rm*(*empty*) $\notin l|_{D'}$ and *noRE*(l') holds. Similar cases hold for other predicates of the four rules of *EPQ*, such as *LEI*, *LI-U*, *matched-C*, *putInSeq*(l, l_1, pri) and *matched-All*. This completes the proof of this lemma. □

C.2 Proof of Lemma 4

Lemma 4. *Given a data-differentiated sequential execution* e*,* $e \in$ *EPQ, if and only if,* $\forall e' \in$ *proj*(e) *and* $\forall R \in$ *last*(e')*, we have* $e' \in$ *MS*(R).

Proof. The *only if* direction is obvious and can be similarly proved as the $EPQ_s \subseteq EPQ$ direction of Lemma 1.

To prove the *if* direction, we proceed as follows: From $e1 = e$, we generate a sequence e_2 as follows:

- If $PQ_3 \in$ *last*(e_1): Then we can see that e_1 contains at least one *rm*(*empty*). e_2 is generated from e_1 by erasing one *rm*(*empty*).
- Else, if $PQ_2^{=} \in$ *last*(e_1): Then we can see that one of the maximal priority of e_1 is unmatched *put* and (possibly) matched *put*. Assume the set of the items of these unmatched *put* is S. e_2 is generated from e_1 by erasing one unmatched *put* which use the item last putted in S.
- Else, if $PQ_2^{>} \in$ *last*(e_1): Then we can see that one of the maximal priority in e_1 is unmatched *put*. Assume the set of the items of these unmatched *put* is S. e_2 is generated from e_1 by erasing one unmatched *put* which use the item last putted in S.
- Else, if $PQ_1^{=} \in$ *last*(e_1): Then we can see that one of the maximal priority in e_1 is of more than one pair of matched *put*. Assume the set of the items of these matched *put* is S. e_2 is generated from e_1 by erasing matched *put* and *rm* of the item which is last putted in S.
- Else, if $PQ_1^{>} \in$ *last*(e_1): Then we can see that one of the maximal priority in e_1 is of one pair of matched *put*. e_2 is generated from e_1 by erasing this pair of matched *put* and *rm*.

Similarly, for each $i > 1$, we obtain e_{i+1} from e_i, until we obtain $e_m = \epsilon$ for some m. It is obvious that $e_m \in$ *EPQ*. For e_{m-1}, since

- $e_m \in$ *EPQ*,
- By assumption, we know that $e_{m-1} \in$ *MS*(R_{m-1}), where $R_{m-1} \in$ *last*(e_{m-1}). This implies that the guard of R_{m-1} is satisfied.

Therefore, we know that $e_{m-1} \in$ *EPQ*. Similarly, we can prove that $e_{m-2}, \ldots, e_1 = e \in$ *EPQ*. □

C.3 Proof of Lemma 5

The prove that *EPQ* is step-by-step linearizability, we investigate each rules individually.

Given a data-differentiated execution and its history, we can abuse notation and mix labels and method events with operations themselves, since items are unique in a data-differentiated execution. For instance, we will reference an operation labeled by $put(p,a)$ as $put(p,a)$.

Given an operation o with call action $cal_o(put,a,p)$ and return actions $ret_o(put)$, its method event is $put(a,p)$. Given an operation o with call action $cal_o(rm)$ and return actions $ret_o(rm,a)$, its method event is $rm(a)$.

Given a data-differentiated execution e and its history h, we can obtain a sequence h' from h by adding $put(a,p)$ (resp., $rm(a)$, $rm(empty)$) between each pair of $cal(put,a,p)$ and $ret(put,a)$ (resp., $cal(rm,a)$ and $ret(rm,a)$, $cal(rm,empty)$ and $ret(rm,empty)$). The projection of h' into method events is called linearization of e and h, and each method event we add in h' can be considered as a linearization point of the corresponding method event. We call such h' an execution with linearization points of e.

Before we prove the step-by-step linearizability of $MS(EPQ_1)$ and $MS(EPQ_2)$, we introduce several lemmas, which are used to ensure some sub-sequences of *EPQ* still belongs to *EPQ*. Given a data-differentiated sequence l and one of its maximal priority *pri*, let $O_c(l,pri)$ and $O_i(l,pri)$ be the set of operations with priorities comparable with *pri* and incomparable with *pri* in l, respectively. Similarly we can define $D_c(l,pri)$ and $D_i(l,pri)$ for items instead of operations. We can see that each priority of items in $O_i(l,pri)$ is either larger or incomparable with priorities of items in $O_c(l,pri)$.

The following lemma shows that if a new sequence is generated by erasing some operations in $O_c(l,pri)$ while keeping the remaining $O_c(l,pri)$ sub-sequences in *EPQ*, then this new sequence is still in *EPQ*. Note that this is different from projection on items.

Lemma 16. *Given a data-differentiated sequential execution $l \in EPQ$ and a maximal priority pri in l, where l does not contain $rm(empty)$. Let l' be generated from l by discarding some operations in $O_c(l,pri)$, and $l'|_{O_c(l,pri)} \in EPQ$. Then, $l' \in EPQ$.*

Proof. Let $l = o_1 \cdot \ldots \cdot o_m$, and $q_0 \xrightarrow{o_1} q_1 \ldots \xrightarrow{o_m} q_m$ be the path of l in LTS_e. Assume that l' is generated from l by discarding $o_{ind1}, \ldots, o_{indn}$. Let D be the set such that D contains a, if $put(a,\_)$ is in $o_{ind1}, \ldots, o_{indn}$. For each i, let q'_i be generated from q_i by erasing items in D.

For each q'_j with $j \neq ind\_\text{-}1$, if o_{j+1} is *put*, then it is obvious that $q'_j \xrightarrow{o_{j+1}} q'_{j+1}$. Else, assume $o_{j+1} = rm(a)$,

- If $rm(a) \in O_c(l,pri)$: By assumption, $l'|_{O_c(l,pri)} \in EPQ$. Therefore, a is in q'_j and is the should-be-removed item in $O_c(l,pri)$. Since each priority of items in $O_i(l,pri)$ is either larger or incomparable with priorities of items in $O_c(l,pri)$, we can removed a_c from q'_j, and then $q'_j \xrightarrow{o_{j+1}} q'_{j+1}$.
- If $rm(a) \in O_i(l,pri)$: By assumption we know that $q_j \xrightarrow{rm(a)} q_{j+1}$. Since q'_j contains the same $D_i(l,pri)$ items as q_j and q'_j contains less $D_c(l,pri)$ items than q_j, we can see that $q'_j \xrightarrow{o_{j+1}} q'_{j+1}$.

For each $q'_{\mathit{indj-1}}$, it is easy to see that $q'_{\mathit{indj-1}} = q'_{\mathit{indj}}$. Therefore, we can see that $l' = o_1 \cdot \ldots o_{\mathit{ind1-1}} \cdot o_{\mathit{ind1+1}} \cdot \ldots \in \mathit{EPQ}$. $\square$

The following lemma shows that if a new sequence is generated by from some time point, erasing operations in $O_i(l, \mathit{pri})$, then this new sequence is still in *EPQ*.

Lemma 17. *Given a data-differentiated sequential execution $l \in \mathit{EPQ}$ and a maximal priority pri in l, where l does not contain rm(empty). Let l' be generated from l by discarding operations in $O_i(l, \mathit{pri})$ from some time point, then, $l' \in \mathit{EPQ}$.*

Proof. Let $l = o_1 \cdot \ldots \cdot o_m$, and $q_0 \xrightarrow{o_1} q_1 \ldots \xrightarrow{o_m} q_m$ be the path of l in LTS_e. Assume that l' is generated from l by discarding all operations o_i if (1) $o_i \in O_i(l, \mathit{pri})$ and (2) $i \geq k$ for a specific index k. Let D be a set such that $a \in D$, if $\mathit{put}(a, \_)$ is in l and not in l'. For each $0 \leq i \leq m$, let q'_i be generated from q_i by erasing items in D.

Let $l' = o'_1 \cdot \ldots \cdot o'_n$, and let f be a function, such that $f(i) = j$, if $o'_i = o_j$.

- We can see that f maps each $0 \leq i \leq k\text{-}1$ into i, and $q'_0 \xrightarrow{o_1} q'_1 \ldots \xrightarrow{o_{k\text{-}1}} q'_{k\text{-}1}$.
- It is easy to see that $q'_{k\text{-}1} = q'_{f(k)-1}$, and for each $i > k$, $q'_{f(k)} = q'_{f(k+1)-1}$.
- If $o_{f(k)}$ is a *put* operation, then it is obvious that $q'_{f(k)-1} \xrightarrow{o_k} q'_{f(k)}$. Else, if $o_{f(k)} = \mathit{rm}(a)$, we can see a is in $q'_{f(k)-1}$, and since (1) $q'_{f(k)-1}$ contains the same $D_c(l, \mathit{pri})$ items as $q_{f(k)-1}$ and $q'_{f(k)-1}$ contains less $D_i(l, \mathit{pri})$ items than $q_{f(k)-1}$, and (2) each priority of items in $O_i(l, \mathit{pri})$ is either larger or incomparable with priorities of items in $O_c(l, \mathit{pri})$, we can see that $q'_{f(k)-1} \xrightarrow{o_k} q'_{f(k)}$. Similarly we can prove the case of $o_{f(j)}$ with $j > k$.

This completes the proof of this lemma. $\square$

The following lemma shows that we can make *put* with maximal priority to happen earlier.

Lemma 18. *Given a data-differentiated sequential execution $l \in \mathit{EPQ}$ and a maximal priority pri in l, where l does not contain rm(empty). Let $l = l_1 \cdot l_2$. Let l_3 be the projection of l_2 into $\{\mathit{put}(\_, \mathit{pri})\}$, and l_4 be the projection of l_2 into other operations. Then, $l' = l_1 \cdot l_3 \cdot l_4 \in \mathit{EPQ}$.*

Proof. Let $l = o_1 \cdot \ldots \cdot o_m$, and $q_0 \xrightarrow{o_1} q_1 \ldots \xrightarrow{o_m} q_m$ be the path of l in LTS_e. Let $l_1 = o_1 \cdot \ldots \cdot o_n$, let $l_2 = o_{n+1} \cdot \ldots \cdot o_m$, let $l_3 = o'_{n+1} \cdot \ldots \cdot o'_k$, let $l_4 = o'_{k+1} \cdot \ldots \cdot o'_m$. Let f be a function, such that $f(i) = j$, if $o'_i = o_j$.

Let q'_i be constructed as follows:

- For $0 \leq i \leq n$, let $q'_i = q_i$.
- For $n{+}1 \leq i \leq k$, let q'_i be obtained from q_n by adding items in $o'_{n+1} \cdot \ldots \cdot o'_i$ with priority *pri* and in the same order.
- For $k{+}1 \leq i \leq m$, let q'_i be obtained from $q_{f(i)}$ by adding items which are (1) with priority *pri*, (2) in $o'_{n+1} \cdot \ldots \cdot o'_i$ and not removed by $o_1 \cdot \ldots \cdot o_{f(i)}$. The order of adding them is the same as $o'_{n+1} \cdot \ldots \cdot o'_i$.

Then, our proof proceeds as follows:

- It is obvious that $q'_0 \xrightarrow{o_1} q'_1 \dots \xrightarrow{o_n} q'_n$ and $q'_n \xrightarrow{o_{n+1}} q'_{n+1} \dots \xrightarrow{o_k} q'_k$.
- For q'_{k+1}: We already know that $q_{f(k+1)-1} \xrightarrow{o_{f(k+1)}} q_{f(k+1)}$, and it is easy to see that $q_{f(k+1)-1}$ is obtained from q_n by adding items in $o_{n+1} \cdot \dots \cdot o_{f(k+1)-1}$.
We can see that q'_k is obtained from q_n by adding items in $o'_{n+1} \cdot \dots \cdot o'_k$, and q'_{k+1} is obtained from $q_{f(k+1)}$ by adding items which are (1) with priority *pri*, (2) in $o'_{n+1} \cdot \dots \cdot o'_i$ and not removed by $o_1 \cdot \dots \cdot o_{f(k+1)}$.
  - If $o_{f(k+1)}$ is an operation of non-*pri* items, then q'_{k+1} is obtained from $q_{f(k+1)}$ by adding items in $o'_{n+1} \cdot \dots \cdot o'_i$. Since non-*pri* priority is either smaller or incomparable with *pri*, we can see that $q'_k \xrightarrow{o_{f(k+1)}} q'_{k+1}$.
  - Otherwise, it is only possible that $o_{f(k+1)} = rm(a)$ for some item a with priority *pri*. We can see that q'_{k+1} is obtained from $q_{f(k+1)}$ by adding items in $o'_{n+1} \cdot \dots \cdot o'_k$ and then remove a. Since $q_{f(k+1)-1} \xrightarrow{o_{f(k+1)}} q_{f(k+1)}$, in $q_{f(k+1)-1}$(and also in q'_k), there is no item with priority less than *pri*, and a is the first-input item of priority *pri*. Therefore, we can see that $q'_k \xrightarrow{o_{f(k+1)}} q'_{k+1}$.
- For q'_{k+i} with $i > 1$: We already know that $q_{f(k+i)-1} \xrightarrow{o_{f(k+i)}} q_{f(k+i)}$, and $q_{f(k+i)-1}$ is obtained from $q_{f(k+i-1)}$ by adding items in $o_{f(k+i-1)+1} \cdot \dots \cdot o_{f(k+i)-1}$.
We can see that q'_{k+i-1} (resp., q'_{k+i}) is obtained from $q_{f(k+i-1)}$ (resp., $q_{f(k+i)}$) by adding items which are (1) with priority *pri*, (2) in $o'_{n+1} \cdot \dots \cdot o'_i$ and not removed by $o_1 \cdot \dots \cdot o_{f(k+i-1)}$ (resp., $o_1 \cdot \dots \cdot o_{f(k+i)}$).
  - If $o_{f(k+i)}$ is an operation of non-*pri* items, then q'_{k+i} is obtained from $q_{f(k+i)}$ by adding items which are (1) with priority *pri*, (2) in $o'_{n+1} \cdot \dots \cdot o'_i$ and not removed by $o_1 \cdot \dots \cdot o_{f(k+i-1)}$. Since non-*pri* priority is either smaller or incomparable with *pri*, we can see that $q'_{k+i-1} \xrightarrow{o_{f(k+i)}} q'_{k+i}$.
  - Otherwise, it is only possible that $o_{f(k+i)} = rm(a)$ for some item a with priority *pri*. We can see that q'_{k+i} is obtained from $q_{f(k+i)}$ by adding items in $o'_{n+1} \cdot \dots \cdot o'_k$ and then remove a. Since $q_{f(k+i)-1} \xrightarrow{o_{f(k+i)}} q_{f(k+i)}$, in $q_{f(k+i)-1}$(and also in q'_{k+i-1}), there is no item with priority less than *pri*, and a is the first-input item of priority *pri*. Therefore, we can see that $q'_{k+i-1} \xrightarrow{o_{f(k+i)}} q'_{k+i}$.

This completes the proof of this lemma. □

The following lemma shows that if a new sequence is generated by make some $O_i(l, pri)$ behaviors to happen earlier, then this new sequence is still in *EPQ*.

Lemma 19. *Given a data-differentiated sequential execution $l \in EPQ$ and a maximal priority pri in l, where l does not contain rm(empty). Let $l|_{O_i(l,pri)} = l_1 \cdot l_2$, let $l' = l_1 \cdot l_3$, where l_3 is the projection of l into non-l_1 operations. Then, $l' \in EPQ$.*

Proof. Let $l = o_1 \cdot \dots \cdot o_m$, and $q_0 \xrightarrow{o_1} q_1 \dots \xrightarrow{o_m} q_m$ be the path of l in LTS_e. Let $l_1 = o'_1 \cdot \dots \cdot o'_n$, let $l_3 = o'_{n+1} \cdot \dots \cdot o'_m$. Let f be a function, such that $f(i) = j$, if $o'_i = o_j$. Let D be the se of items which are added and not removed in $o'_1 \cdot \dots \cdot o'_n$.

Let q'_i be constructed as follows:

- It is easy to see that $l_1 \in EPQ$, and let $q'_0 \xrightarrow{o'_1} q'_1 \dots \xrightarrow{o'_n} q'_n$ be the path of l_1 in LTS_e.

- For n+$1 \leq i \leq m$, let q'_i be obtained from $q_{f(i)}$ by adding items in D. The order of adding them is the same as $o'_1 \cdot \ldots \cdot o'_n$.

Then, our proof proceeds as follows:

- We already know that $q'_0 \xrightarrow{o'_1} q'_1 \ldots \xrightarrow{o'_n} q'_n$.
- For q'_{n+i}: We already know that $q_{f(n+i)-1} \xrightarrow{o_{f(n+i)}} q_{f(n+i)}$, and it is easy to see that $q_{f(n+i)-1}$ is obtained from $q_{f(n+i-1)}$ by adding D-items in $o_{f(n+i-1)+1} \cdot \ldots \cdot q_{f(n+i)-1}$.
  We can see that q'_{n+i-1} (resp., q'_{n+i}) is obtained from $q_{f(n+i-1)}$ (resp., $q_{f(n+i)}$) by adding remanning items in D.
  - If $o_{f(n+1)}$ is an operation of $O_c(l, pri)$ items, since priority in $O_i(l, pri)$ is either larger or incomparable with priority in $O_c(l, pri)$, we can see that $q'_n \xrightarrow{o_{f(n+1)}} q'_{n+1}$.
  - Otherwise, it is only possible that $o_{f(k+1)} = rm(a)$ for some item a in $O_c(l, pri)$. Since $q_{f(n+i)-1} \xrightarrow{o_{f(n+i)}} q_{f(n+i)}$, in $q_{f(n+i)-1}$(and also in q'_{n+i-1}), there is no item with priority less than *pri*, and a is the first-input item of priority *pri*. Therefore, we can see that $q'_{n+i-1} \xrightarrow{o_{f(n+i)}} q'_{n+i}$.

This completes the proof of this lemma. □

The following lemma shows that if a new sequence is generated by replacing a prefix with another one which make the priority queue has same content, then this new sequence is still in *EPQ*.

Lemma 20. *Given a data-differentiated sequential execution $l \in$ EPQ. Let $l = l_1 \cdot l_2$. Given $l_3 \in$ EPQ. Assume that the priority queue has same content after executing l_1 and l_3. Let $l' = l_3 \cdot l_2$. Then, $l' \in$ EPQ.*

Proof. Let $l = o_1 \cdot \ldots \cdot o_m$ and let $q_0 \xrightarrow{o_1} q_1 \ldots \xrightarrow{o_m} q_m$ be the path of l in LTS_e. Let $l_1 = o_1 \cdot \ldots \cdot l_k$, $l_3 = o'_1 \cdot \ldots \cdot o'_n$ and let $q_0 \xrightarrow{o'_1} q'_1 \ldots \xrightarrow{o'_n} q'_n$ be the path of l_3 in LTS_e.

By assumption we know that $q_k = q'_n$. Then it is not hard to see that $q_0 \xrightarrow{o'_1} q'_1 \ldots \xrightarrow{o'_n} q'_n \xrightarrow{o_{k+1}} q_{k+1} \ldots \xrightarrow{o_m} q_m$ is a path in LTS_e, and then $l' \in EPQ_s$. By Lemma 1, we know that $l' \in$ *EPQ*. □

With the help of Lemma 16, Lemma 17, Lemma 18, Lemma 19 and Lemma 20, we can now prove that EPQ_1 is step-by-step linearizability.

Lemma 21. *If a data-differentiated concurrent execution e is linearizable w.r.t. MS(EPQ_1) with witness x, then $e \setminus x \sqsubseteq$ EPQ $\Rightarrow e \sqsubseteq$ EPQ.*

Proof. Let h be the data-differentiated history of e, and l be an sequential execution such that $h \sqsubseteq l$ and l matches EPQ_1 with witness x. Let the priority of x be pri_x, and let $h' = h \setminus x$ and assume that $h' \sqsubseteq l' \in$ *EPQ*. Let e_{lp} be an execution with linearization points of e and the linearization points is added according to l'. Or we can say, e_{lp} is generated from e by instrumenting linearization points, and the projection of e_{lp} into method event is l'.

According to $MS(EPQ_1)$, there exist sequences u, v, and w, such that $l = u \cdot put(x, pri_x) \cdot v \cdot rm(x) \cdot w$ and u, v, w, x and pri_x satisfy the guard of EPQ_1. Let l'_v be the shortest prefix of l' that contains all method event of $u \cdot v$.

Let U, V and W be the set of operations of u, v and w, respectively. Let us change $O_i(l, pri_x)$ elements in U, V and W, while keep $O_c(l, pri_x)$ elements unchanged. We proceed by several loops: In the first loop, we start from the first $O_i(l, pri_x)$-element of l'_v, and let it be o_h,

- Case 1: If in e_{lp}, the linearization point of o_h is before $ret(rm, x)$, and no $O_c(l, pri_x)$-element in W happens before o_h. Then, o_h is in the new version of $U \cup V$.
- Case 2: Else, if in e_{lp}, the linearization point of o_h is before $ret(rm, x)$, and there exists $O_c(l, pri_x)$-element o_w in W, such that $o_w <_{hb} o_h$. Then, in l'_v, we put o_h and all $O_i(l, pri_x)$-element whose linearization points is after the linearization point of o_h into new version of W, and then stop the process of changing U, V and W.
- Case 3: Else, if in e_{lp}, the linearization point of o_h is after $ret(rm, x)$. Then, in l'_v, we put o_h and all $O_i(l, pri_x)$-element whose linearization points is after the linearization point of o_h into new version of W, and then stop the process of changing U, V and W.

In the next loop, we consider the second $O_i(l, pri_x)$-element of l'_v, and so on. Our process proceed, until either all element in l'_v are in new version of $U \cup V$, or case 2 or case 3 happens and this process terminates. Let $U' \cup V'$ and W' be the new version of $U \cup V$ and W after the process terminates, respectively. Let O_+ be the set of operations that are moved into $U' \cup V'$ in the process, and let O_- be the set of operations that are moved into W' in the process.

Let $l'_{u'v'}$ be the projection of l' into $U' \cup V'$, let O_x be the set of $put(\_, pri_x)$ while the item is not x in h. Let l''_a be the longest prefix of $l'_{u'v'}$, where linearization of each operation of l''_a is before $ret(put, b)$ in e_{lp}. Let l''_d be the projection of l' into operations of O_x that are not in l''_a. Let $l''_1 = l''_a \cdot l''_d$. Let l''_2 be the projection of l' into operations of $l'_{u'v'}$ that are not in l''_1. Let l''_3 be the projection of l' into operations which are not in $l'_{u'v'}$. Let $l'' = l''_1 \cdot put(x, pri_x) \cdot l''_2 \cdot rm(x) \cdot l''_3$.

To prove $h \sqsubseteq l''$, we define a graph G whose nodes are the operations of h and there is an edge from operation o_1 to o_2, if one of the following case holds

- o_1 happens-before o_2 in h,
- the method event corresponding to o_1 in l'' is before the one corresponding to o_2.

Assume there is a cycle in G. According the the property of interval order and the fact that the order of l'' is total, we know that there must exists o_1 and o_2, such that o_1 happens-before o_2 in h, but the corresponding method events are in the opposite order in l''. Then, we consider all possible case of o_1 and o_2 as follows: Let O_a and O_d be the set of operations in l''_a and l''_d, respectively.

- If $o_2 \in l''_1 \wedge o_1 \in l''_1$:

- If $o_1, o_2 \in O_a$ or $o_1, o_2 \in O_d$: Then l' contradicts with happen before relation of h.
- If $o_2 \in O_a \wedge o_1 \in O_d$: Then the order of linearization points of e_{lp} contradicts with happen before relation of h.

- If $o_2 \in l''_1 \wedge o_1 = put(x, pri_x)$:
  - If $o_2 \in O_a$: This is impossible, since the linearization point of operations in O_a is before $ret(put, b)$ in e_{lp}.
  - If $o_2 \in O_d$: Then l contradicts with happen before relation of h.
- If $o_2 \in l''_1 \wedge o_1 \in l''_2$:
  - If $o_2 \in O_a$: This violates the order of linearization point in e_{lp}.
  - If $o_2 \in O_d$: According to l, we can see that $put(x, pri_x)$ does not happen before any operation in O_x. Then we can see that the linearization point of o_1 is before $ret(put, x)$ and $o_1 \in O_a$. This violates that $o_1 \in l''_2$.
- If $o_2 \in l''_1 \wedge o_1 = rm(x)$:
  - If $o_2 \in U \cup V$: Then l contradicts with happen before relation of h.
  - If $o_2 \in O_+$: This is impossible, since the linearization point of operations in O_+ is before $ret(rm, x)$ in e_{lp}.
- If $o_2 \in l''_1 \wedge o_1 \in l''_3$:
  - If $o_1 \in W \wedge o_2 \in U \cup V$: Then l contradicts with happen before relation of h.
  - If $o_1 \in W \wedge o_2 \in O_+$:
    - If $o_1 \in O_i(l, pri_x)$: Then according to the construction process of $U' \cup V'$ and W', we can see that $o_1 \in O_+$ and then $o_1 \in U' \cup V'$, which contradicts that $o_1 \in l''_3$.
    - If $o_1 \in O_c(l, pri_x)$: Then according to the construction process of $U' \cup V'$ and W', we can see that $o_2 \in O_-$, which contradicts that $o_2 \in O_+$.
  - If $o_1 \in O_- \wedge o_2 \in U \cup V$:
    - If the reason of $o_1 \in O_-$ is case 2: Let o_h be as in case 2. Then there exists $O_c(l, pri_x)$-element $o_w \in W$, and in e_{lp}, $ret(o_w)$ is before $cal(o_h)$, the linearization point of o_h is before the linearization point of o_1, and $ret(o_1)$ is before $cal(o_2)$. Therefore, we can see that $o_w <_{hb} o_2$, and then l contradicts with happen before relation of h.
    - If the reason of $o_1 \in O_-$ is case 3: Let o_h be as in case 3. Then in e_{lp}, $ret(rm, x)$ is before the linearization point of o_h, the linearization point of o_h is before the linearization point of o_1, and $ret(o_1)$ is before $cal(o_2)$. Therefore, we can see that $rm(x) <_{hb} o_2$, and then l contradicts with happen before relation of h.
  - If $o_1 \in O_- \wedge O_2 \in O_+$: This is impossible, since in e_{lp}, the linearization points of operations in O_+ is before the linearization points of operations in O_-.
- If $o_2 = put(x, pri_x) \wedge o_1 \in l''_2$: This is impossible, since in e_{lp}, the linearization points of operations in l''_2 is after $ret(put, x)$.
- If $o_2 = put(x, pri_x) \wedge o_1 = rm(x)$: Then l contradicts with happen before relation of h.
- If $o_2 = put(x, pri_x) \wedge o_1 \in l''_3$:
  - If $o_1 \in W$: Then l contradicts with happen before relation of h.
  - If $o_1 \in O_-$:

- If the reason of $o_1 \in O_-$ is case 2: Let o_h be as in case 2. Then there exists $O_c(l, pri_x)$-element $o_w \in W$, and in e_{lp}, $ret(o_w)$ is before $cal(o_h)$, the linearization point of o_h is before the linearization point of o_1, and $ret(o_1)$ is before $cal(put, x, pri_x)$. Therefore, we can see that $o_w <_{hb} put(x, pri_x)$, and then l contradicts with happen before relation of h.
- If the reason of $o_1 \in O_-$ is case 3: Let o_h be as in case 3. Then in e_{lp}, $ret(rm, x)$ is before the linearization point of o_h, the linearization point of o_h is before the linearization point of o_1, and $ret(o_1)$ is before $cal(put, x, pri_x)$. Therefore, we can see that $rm(x) <_{hb} put(x, pri_x)$, and then l contradicts with happen before relation of h.

- If $o_2 \in l''_2 \wedge o_1 \in l''_2$: Then l' contradicts with happen before relation of h.
- If $o_2 \in l''_2 \wedge o_1 = rm(x)$: We can prove this similarly as the case of $o_2 \in l''_1 \wedge o_1 = rm(x)$.
- If $o_2 \in l''_2 \wedge o_1 \in l''_3$: We can prove this similarly as the case of $o_2 \in l''_1 \wedge o_1 \in l''_3$.
- If $o_2 = rm(x) \wedge o_1 \in l''_3$:
  - If $o_1 \in W$: Then l contradicts with happen before relation of h.
  - If $o_1 \in O_-$:
    - If the reason of $o_1 \in O_-$ is case 2: Let o_h be as in case 2. Then there exists $O_c(l, pri_x)$-element $o_w \in W$, and in e_{lp}, $ret(o_w)$ is before $cal(o_h)$, the linearization point of o_h is before the linearization point of o_1.
      Since l is consistent with the happen before order of h, we can see that $cal(rm, x)$ is before $ret(o_w)$. Therefore, we can see that the linearization point of o_1 is after $cal(rm, x)$, and then it is impossible that $o_1 <_{hb} rm(x)$.
    - If the reason of $o_1 \in O_-$ is case 3: Let o_h be as in case 3. Then in e_{lp}, $ret(rm, x)$ is before the linearization point of o_h, and the linearization point of o_h is before the linearization point of o_1. Therefore, we can see that the linearization point of o_1 is after $ret(rm, x)$, and then it is impossible that $o_1 <_{hb} rm(x)$.
- If $o_2 \in l''_3 \wedge o_1 \in l''_3$: Then l' contradicts with happen before relation of h.

Therefore, we know that G is acyclic, and then we know that $h \sqsubseteq l''$.
It remains to prove that $l'' \in EPQ$. The process for proving $l'' \in EPQ$ is as follows:

- Since $l' \in EPQ$ and l'_v is a prefix of l', it is obvious that $l'_v \in EPQ$.
- $l'_{u'v'}$ can be obtained from l'_v as follows:
  - Discard $O_c(l, pri_x)$-element that are in W and keep $O_c(l, pri_x)$-element in $U \cup V$ unchanged.
  - From some time point, discard all the $O_i(l, pri_x)$ operations after this time point.

  From *matched-C*$(u \cdot v, pri_x)$, we can see that $O_c(l, pri_x)$-element in $U \cdot V$ is matched, and then it is easy to see that the projection of l'_v in to $O_c(l, pri_x)$-element in $U \cdot V$ is still in EPQ. By Lemma 16 and Lemma 17, we can see that $l'_{u'v'} \in EPQ$.
- $l''_1 \cdot l''_2$ can be obtained from $l'_{u'v'}$ as follows: Execute until reaching some time point t, then first execute all O_x operations after t, and then execute remanning operations. By Lemma 18, we can see that $l''_1 \cdot l''_2 \in EPQ$.

- Let l'_e be obtained from l' by discarding $O_c(l, pri_x)$-element in $U \cdot V$. By Lemma 3, we can see that $l'_e \in$ *EPQ*.
- Let $l'_e|_{O_i(l,pri_x)} = l'_f \cdot l'_g$, where l'_f is the projection of $O_i(l, pri_x)$-element in $U \cdot V$. Let l'_h be obtained from l'_e by discarding operations in l'_f. By Lemma 19, we can see that $l'_f \cdot l'_h \in$ *EPQ*.
- By Lemma 3, it is obvious that $l'_f \in$ *EPQ*. From *matched-C*$(u \cdot v, pri_x)$, we can see that the content of priority queue after executing $l''_1 \cdot l''_2$ is the same as after executing l'_f. By Lemma 20, we can see that $l''_1 \cdot l''_2 \cdot l'_h \in$ *EPQ*. It is easy to see that $l'_h = l''_3$, and then $l''_1 \cdot l''_2 \cdot l''_3 \in$ *EPQ*.
- Since u, v, w, x and pri_x satisfy the guard of *EPQ*$_1$, it is easy to see that $l'' = l''_1 \cdot put(x, pri_x) \cdot l''_2 \cdot rm(x) \cdot l''_3 \in$ *EPQ*.

Therefore, we prove that $h \sqsubseteq l'' \in$ *EPQ*. This completes the proof of this lemma. □

Lemma 22. *If a data-differentiated concurrent execution* e *is linearizable w.r.t. MS*(EPQ_2) *with witness* x*, then* $e \setminus x \sqsubseteq$ *EPQ* $\Rightarrow e \sqsubseteq$ *EPQ.*

Proof. Let h be the data-differentiated history of e, and l be an sequential execution such that $h \sqsubseteq l$ and l matches *EPQ*$_2$ with witness x. Let the priority of x be pri_x, and let $h' = h \setminus x$ and assume that $h' \sqsubseteq l' \in$ *EPQ*. Let e_{lp} be an execution with linearization points of e and the linearization points is added according to l'. Or we can say, e_{lp} is generated from e by instrumenting linearization points, and the projection of e_{lp} into method event is l'.

According to *MS*(EPQ_2), there exist sequences u and v, such that $l = u \cdot put(x, pri_x) \cdot v$ and u, v, x and pri_x satisfy the guard of *EPQ*$_2$.

Let l''_a be the longest prefix of l' such that linearization point of each operation of l''_a is before $ret(put, x)$ in e_{lp}. Let O_x be the set of $put(\_, pri_x)$ while the item is not x in h. Let l''_s be the projection of l' into operations of O_x that are not in l''_a. Let $l''_1 = l''_a \cdot l''_s$. Let l''_2 be the projection of l' into operations of l' that are not in l''_1. Let $l'' = l''_1 \cdot put(x, pri_x) \cdot l''_2$.

To prove $h \sqsubseteq l''$, we define graph G as in Lemma 21. Assume that there is a cycle in G, then there must exists o_1 and o_2, such that o_1 happens-before o_2 in h, but the corresponding method events are in the opposite order in l''. Then, we consider all possible case of o_1 and o_2 as follows: Let O_a and O_s be the set of operations in l''_a and l''_s, respectively.

- If $o_2 \in l''_1 \wedge o_1 \in l''_1$:
  - If $o_1, o_2 \in O_a$ or $o_1, o_2 \in O_s$: Then l' contradicts with happen before relation of h.
  - If $o_2 \in O_a \wedge o_1 \in O_s$: It is not hard to see that $put(x, pri_x) <_{hb} o_2$. Then, it is impossible to locate the linearization point of o_2 before $ret(put, x)$ in e_{lp}.
- If $o_2 \in l''_1 \wedge o_1 = put(x, pri_x)$:
  - If $o_2 \in O_a$: This is impossible, since in e_{lp}, the linearization point of o_1 is before $ret(put, x)$.

- If $o_2 \in O_s$: This is impossible, since $o_2 = \mathit{put}(\_, \mathit{pri}_x)$, and l is consistent with happen before relation of h.

- If $o_2 \in l''_1 \wedge o_1 \in l''_2$:
  - If $o_2 \in O_a$: This is impossible, since in e_{lp}, the linearization point of operation in l''_a is before the linearization point of operations in l''_2.
  - If $o_2 \in O_s$: Since no $\mathit{put}(\_, \mathit{pri}_x)$ happens before $\mathit{put}(x, \mathit{pri}_x)$ in h, $\mathit{cal}(o_2)$ is before $\mathit{ret}(\mathit{put}, x)$. Since $o_1 <_{hb} o_2$, we can see that $\mathit{ret}(o_1)$ is before $\mathit{cal}(o_2)$, and then $\mathit{ret}(o_1)$ is before $\mathit{ret}(\mathit{put}, x)$. Then the linearization point of o_1 can only be before $\mathit{ret}(\mathit{put}, x)$, and $o_1 \in l''_a$, which contradicts that $o_1 \in l''_2$.
- If $o_2 = \mathit{put}(x, \mathit{pri}_x) \wedge o_1 \in l''_2$: Then since the linearization point of o_1 can only be before $\mathit{ret}(\mathit{put}, x)$, we can see that $o_1 \in l''_a$, which contradicts that $o_1 \in l''_2$.
- If $o_2 \in l''_2 \wedge o_1 \in l''_2$: Then l' contradicts with happen before relation of h.

Therefore, we know that G is acyclic, and then we know that $h \sqsubseteq l''$.

It remains to prove that $l'' \in \mathit{EPQ}$. $l''_a \cdot l''_s \cdot l''_2$ can be obtained from l' as follows: Execute until reaching some time point t, then first execute all O_x operations after t, and then execute remanning operations. By Lemma 18, we can see that $l''_1 \cdot l''_2 = l''_a \cdot l''_s \cdot l''_2 \in \mathit{EPQ}$. Since u, v, x and pri_x satisfy the guard of EPQ_2, it is easy to see that $l'' = l''_1 \cdot \mathit{put}(x, \mathit{pri}_x) \cdot l''_2 \in \mathit{EPQ}$.

Therefore, we prove that $h \sqsubseteq l'' \in \mathit{EPQ}$. This completes the proof of this lemma.

$\square$

Lemma 23. *If a data-differentiated concurrent execution* e *is linearizable w.r.t.* $\mathit{MS}(\mathit{EPQ}_3)$ *and* o *is a* $\mathit{rm}(\mathit{empty})$ *event, then* $e \setminus o \sqsubseteq \mathit{PQueue} \Rightarrow e \sqsubseteq \mathit{PQueue}$.

Proof. Let h be the data-differentiated history of e, l be an sequential execution such that $h \sqsubseteq l$, l matches EPQ_3 and o is a *rm* method event in h. Let $h' = h \setminus o$ and assume that $h' \sqsubseteq l' \in \mathit{EPQ}$.

According to EPQ_3, there exist sequences u and v, such that $l = u \cdot \mathit{rm}(\mathit{empty}) \cdot v$, where all the *put* operations and *rm* in u are matched.

Let E_L be the set of method events in u and E_R be the set of method events in v. Let $l'_L = l'|_{E_L}$ and $l'_R = l'|_{E_R}$. Let sequence $l'' = l'_L \cdot o \cdot L'_R$. Since priority queue is closed under projection (Lemma 3) and all the *put* operations and *rm* in u are matched, we know that $l'_L \in \mathit{EPQ}$ and the the priority queue is empty after executing l'_L. Then we know that $l'_L \cdot \mathit{rm}(\mathit{empty}) \in \mathit{EPQ}$. Since l'_R is obtained from l' by discarding pairs of matched *put* and *rm* operations, it is easy to see that $L'_R \in \mathit{EPQ}$, and then we know that $l'' = l'_L \cdot o \cdot L'_R \in \mathit{EPQ}$.

It remains to prove that $h \sqsubseteq l''$. To prove $h \sqsubseteq l''$, we define graph G as in Lemma 21. Assume that there is a cycle in G, then there must exists o_1 and o_2, such that o_1 happens-before o_2 in h, but the corresponding method events are in the opposite order in l''. Then, we consider all possible case of o_1 and o_2 as follows:

- $o_1, o_2 \in l'_L$, or $o_1, o_2 \in l'_R$: Then l' contradicts with happen before relation of h.
- If $o_1 = o \wedge o_2 \in l'_L$, or $o_1 \in l'_R \wedge o_2 \in l'_L$, or $o_1 \in l'_R \wedge o_2 = o$, then l contradicts with happen before relation of h.

Therefore, we know that G is acyclic, and then we know that $h \sqsubseteq$ *EPQ*. $\square$

The following lemma states that *EPQ* is step-by-step linearizability, it is a direct consequence of Lemma 21, Lemma 22 and Lemma 23.

Lemma 5. *EPQ is step-by-step linearizability.*

Proof. This is a direct consequence of Lemma 21, Lemma 22 and Lemma 23. $\square$

C.4 Proof of Lemma 6

Lemma 6. *Given a data-differentiated execution e, $e \sqsubseteq$ EPQ, if and only if, $\forall e' \in$ proj(e) and $\forall R \in$ last(e'), we have $e' \sqsubseteq$ MS(R).*

Proof. To prove the *only if* direction, assume that $e \sqsubseteq l \in$ *EPQ*. Given $e' = e|_D$ and $l' = l|_D$, it is easy to see that $e' \sqsubseteq l'$, and by Lemma 3, we can see that $l' \in$ *EPQ*. Then by Lemma 4 we know that for each $R \in$ *last*(l'), we have $l' \in$ *MS*(R).

To prove the *if* direction, given $e_1 = e$, we generate sequence e_2 from e_1 as follows: Since $e_1 \in$ *proj*(e), we know that for each $R_1 \in$ *last*(e_1), we have $e_1 \sqsubseteq$ *MS*(R_1). We choose an arbitrary R_1 in *last*(e_1),

- If $R_1 = EPQ_3$: e_2 is generated from e_1 by erasing call and return of one *rm*(*empty*) operation.
- Else, if $R_1 = EPQ_2^{=}, EPQ_2^{>}, EPQ_1^{=}, EPQ_1^{>}$, and the witness is x: e_2 is generated from e_1 by erasing call and return of method event of item x.

Similarly, for each $i > 1$, we obtain e_{i+1} from e_i, until we obtain $e_m = \epsilon$ for some m. It is obvious that $e_m \sqsubseteq$ *EPQ*. For e_{m-1}, since

- $e_m \sqsubseteq$ *EPQ*,
- If *last*$(e_{m-1}) = R_{m-1} \in \{EPQ_1^{>}, EPQ_1^{=}, EPQ_2^{>}, EPQ_2^{=}\}$ and e_{m-1} matches R_{m-1} with witness x: We already know that $e_{m-1} \sqsubseteq$ *MS*(R_{m-1}), $e_m = e_{m-1} \setminus x \sqsubseteq$ *EPQ*, and by step-by-step linearizability of *EPQ* (Lemma 5), we can see that $e_{m-1} \sqsubseteq$ *EPQ*.
- If *last*$(e_{m-1}) = R_{m-1} = EPQ_3$ and o is a *rm*(*empty*) in e_{m-1} that is removed in the process of constructing e_m: We already know that $e_{m-1} \sqsubseteq$ *MS*(R_{m-1}), $e_m = e_{m-1} \setminus o \sqsubseteq$ *EPQ*, and by step-by-step linearizability of *EPQ* (Lemma 5), we can see that $e_{m-1} \sqsubseteq$ *EPQ*.

Therefore, we know that $e_{m-1} \in$ *EPQ*. Similarly, we can prove that $e_{m-2}, \ldots, e_1 = e \in$ *EPQ*. $\square$

D Proofs and Definitions in Section 6

D.1 Proofs and Definitions in Subsection 6.1

[1] states that, given a differentiated queue execution e without *deq*(*empty*), e is not linearizable with respect to queue, if one of the following cases holds for some a, b: (1)

deq(b) $<_{hb}$ *enq*(b), (2) there are are no *enq*(b) and at least one *deq*(b), (3) there are are one *enq*(b) and more than one *deq*(b), and (4) *enq*(a) $<_{hb}$ *enq*(b), and *deq*(b) $<_{hb}$ *deq*(a), or *deq*(a) does not exists.

For each such case, we construct a witness automata. We generate witness automata $\mathcal{A}^1_{SinPri}$ for the first case, and it is shown in Fig. 10. Here $c_1 = cal(put, a, anyPri)$, $ret(put, a), cal(rm, a), ret(rm, a), cal(rm, b), cal(rm, empty), ret(rm, empty)$, $c_2 = c_1 + ret(rm, b)$, $c_3 = c_2 + ret(put, b)$.

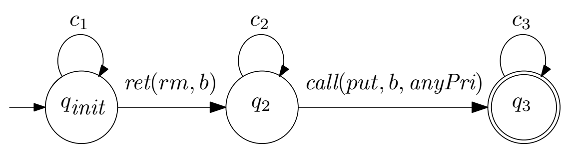

Fig. 10. Automaton $\mathcal{A}^1_{SinPri}$

We generate witness automata $\mathcal{A}^2_{SinPri}$ for the second case, and it is shown in Fig. 11. Here $c_1 = cal(put, a, anyPri), ret(put, a), cal(rm, a), ret(rm, a), cal(rm, empty), ret(rm, empty)$, $c_2 = c_1 + cal(rm, b) + ret(rm, b)$.

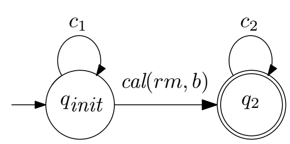

Fig. 11. Automaton $\mathcal{A}^2_{SinPri}$

We generate witness automata $\mathcal{A}^3_{SinPri}$ for the third case, and it is shown in Fig. 12. Here $c_1 = cal(put, a, anyPri), ret(put, a), cal(rm, a), ret(rm, a), cal(rm, empty), ret(rm, empty)$, $c_2 = c_1 + ret(put, b)$, $c_3 = c_2 + ret(rm, b)$, $c_4 = c_3 + cal(rm, b)$, $c_5 = c_1 + ret(rm, b)$, $c_6 = c_5 + cal(rm, b)$.

We generate witness automata $\mathcal{A}^4_{SinPri}$ for the forth case, and it is shown in Fig. 13. Here $c_1 = c + cal(rm, b)$, and $c_2 = c + ret(put, b) + cal(rm, a) + ret(rm, a)$, where $c = cal(put, d, anyPri), ret(put, d), cal(rm, d), ret(rm, d), cal(rm, empty), ret(rm, empty)$.

Let $Auts_{sinPri} = \{\mathcal{A}^1_{SinPri}, \mathcal{A}^2_{SinPri}, \mathcal{A}^3_{SinPri}, \mathcal{A}^4_{SinPri}\}$. Let us prove Lemma 7.

Lemma 7. *Given a data-independent implementations $\mathcal{I}$ of extended priority queue, $\mathcal{I} \cap Auts_{sinPri} \neq \emptyset$, if and only if there exists $e \in \mathcal{I}_{\neq}$, $e' \in proj(e)$, such that e' is single-priority without rm(empty), and transToQueue(e') does not linearizable to queue.*

Proof. [1] states that, given a differentiated queue execution e without *deq*$(empty)$, e is not linearizable with respect to queue, if one of the following cases holds for some v_a, v_b: (1) *deq*(v_b) $<_{hb}$ *enq*(v_b), (2) there are are no *enq*(v_b) and at least one *deq*(v_b), (3) there are are one *enq*(v_b) and more than one *deq*(v_b), and (4) *enq*(v_a) $<_{hb}$ *enq*(v_b), and *deq*(v_b) $<_{hb}$ *deq*(v_a), or *deq*(v_a) does not exists.

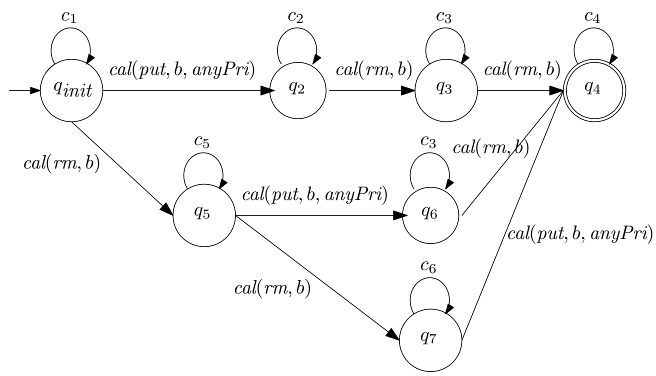

Fig. 12. Automaton $\mathcal{A}^3_{SinPri}$

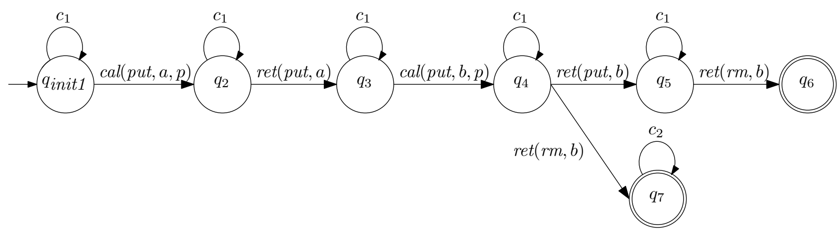

Fig. 13. Automaton $\mathcal{A}^4_{SinPri}$

Let us prove the *only if* direction. Assume that there exists execution $e_0 \in \mathcal{I}$ and e_0 is accepted by an automaton in $Auts_{sinPri}$. By data-independence, we can see that there exists a data-differentiated $e \in \mathcal{I}$ and renaming function, such that $e_0 = r(e)$. Let e' be obtained from e by first removing $rm(empty)$, and then,

- If e_0 is accepted by $\mathcal{A}^1_{SinPri}$, $\mathcal{A}^2_{SinPri}$ or $\mathcal{A}^3_{SinPri}$: Then remove all items that are not renamed into b by r.
- If e_0 is accepted by $\mathcal{A}^4_{SinPri}$: Then remove all items that are not renamed into a or b by r.

It is obvious that $e' \in proj(e)$. It is easy to see that $transToQueue(e')$ satisfies one of above conditions, and then $transToQueue(e')$ is not linearizable w.r.t queue.

Let us prove the *if* direction. Assume that exists $e \in \mathcal{I}_{\neq}$, $e' \in proj(e)$, such that e' is single-priority without $rm(empty)$, and $transToQueue(e')$ does not linearizable to queue. Then we construct a renaming function r as follows:

- If this is because case 1, case 2 or case 3: r maps v_b into b and maps all other values into a.
- If this is because case 4: r maps v_a and v_b into a and b, respectively, and maps all other values into d.

Then it is not hard to see that $r(e) \in \mathcal{I}$ and it is accepted by some automaton in $Auts_{sinPri}$. This completes the proof of this lemma. $\square$

D.2 Proofs and Definitions in Subsection 6.2

The following lemma states that, from linearization of sub-histories, we can merge them and obtain a linearization (regardless of whether it belongs to sequential specification) of the whole history.

Lemma 24. *Given a history h, operation sets S_1, S_2 and sequences l_1 and l_2. Let $h_1 = h|S_1$ and $h_2 = h|S_2$. Assume that $h_1 \sqsubseteq l_1$, $h_2 \sqsubseteq l_2$, and $S_1 \cup S_2$ contains all operations of h. Then, there exists a sequence l, such that $h \sqsubseteq l$, $l|S_1 = l_1$ and $l|S_2 = l_2$.*

Proof. Given a history h and a operation $o \in S_2$, let $\mathit{MB}(o) = \{o' | o' \in S_1$ and o' happens before o in $h\}$, let $\mathit{SBI}(o) = \mathit{min}\{i | l_1[0,i]$ contains all elements of $\mathit{MB}(o)\}$.

Let $l = s_1 \cdot l_2[1] \cdot \ldots \cdot l_2[n] \cdot s_{n+1}$ be generated as follows, where $n = |l_2|$:

- $s_1 = l_1[0, \mathit{SBI}(l_2[1])]$,
- If $s_1 \cdot l_2[1] \cdot \ldots \cdot l_2[i]$ already contains $l_1(\mathit{SBI}(l_2[i{+}1]))$, then $s_{i+1} = \epsilon$. Otherwise, s_{i+1} is a subsequence of l_1, which starts from the next of last elements of $s_1 \cdot l_2[1] \cdot \ldots \cdot l_2[i]$ in l_1 and ends in $l_1(\mathit{SBI}(l_2[i{+}1]))$.

It is obvious that $l|S_1 = l_1$ and $l|S_2 = l_2$, and it remains to prove that $h \sqsubseteq l$. We prove this by contradiction. Assume that h is not linearizable with respect to l. Then there must be two operations of h, such that $o_1 <_{hb} o_2$ in h but o_2 before 0_1 in l. Since $l|S_1 = l_1$, $l|S_2 = l_2$, and $h_1 \sqsubseteq l_1$, $h_2 \sqsubseteq l_2$, it is easy to see that it is impossible that $o_1, o_2 \in S_1$ or $o_1, o_2 \in S_2$. There are only two possibilities:

- $o_1 \in S_1 \wedge o_2 \in S_2$. Then we can see that $o_2 = l_2[i]$ and $o_1 \in s_j$ for some $i < j$. Since $o_1 <_{hb} o_2$, we know that $o_1 \in \mathit{SBI}(o_2)$. By the construction of l, we know that o_1 must be in s_k for some $k \leq i$, contradicts that $o_1 \in s_j$ with $i < j$.
- $o_1 \in S_2 \wedge o_2 \in S_1$. Then we can see that $o_2 \in s_i$ and $o_1 = l_2[j]$ for some $i \leq j$. It is easy to see that this leads to contradiction when $i = j$. For the case of $i \neq j$, we need to satisfy the following requirements: (1) o_1 ($l_2[j]$) does not happen before $l_2[i]$, (2) $l_2[i]$ does not happen before o_2, (3) o_2 is either overlap or happens before $o' \in \mathit{MB}(l_2[i])$, and (4) $o' <_{hb} l_2[i]$. By enumeration we can see that it is impossible that above four conditions be satisfied while $o_1 <_{hb} o_2$.

This completes the proof of this lemma. □

With Lemma 24, we can now prove Lemma 8.

Lemma 8. *Given a data-differentiated execution e and $R = \mathit{EPQ}_1^{>}$ (resp., $R = \mathit{EPQ}_1^{=}$, $R = \mathit{EPQ}_2^{>}$, or $R = \mathit{EPQ}_2^{=}$). $e \sqsubseteq \mathit{MS}(R)$ with witness x, if and only if pri-$\mathit{Exec}(e) \sqsubseteq \mathit{MS}(R)$ with witness x, where pri is the priority of x.*

Proof. We deal with the case of $R = \mathit{EPQ}_1^{>}$, and other cases can be similarly dealt with.

To prove the *only if* direction, given $e \sqsubseteq \mathit{MS}(\mathit{EPQ}_1^{>})$ and such *pri* and x. Since $e \sqsubseteq \mathit{MS}(\mathit{EPQ}_1^{>})$ with witness x, we know that $e \sqsubseteq u \cdot \mathit{put}(x, \mathit{pri}) \cdot v \cdot \mathit{rm}(x) \cdot w$, where u, v, w, x and *pri* satisfy the guard of $EPQ_1^>$. Let u', v' and w' be obtained from u, v and w by erasing all items with priority incomparable with *pri*, respectively. It is not hard to see that u', v', w', x and *pri* satisfy the guard of $EPQ_1^>$, and then $e \sqsubseteq l = u' \cdot put(x,pri) \cdot v' \cdot rm(x) \cdot w' \in MS(EPQ_1^>)$.

To prove the *if* direction, given $e' = pri\text{-}Exec(e)$ and such x and *pri*. Since $e' \sqsubseteq MS(EPQ_1^>)$ with witness x, we know that $e' \sqsubseteq l_1 = u \cdot put(x,pri) \cdot v \cdot rm(x) \cdot w$, where u, v, w, x and *pri* satisfy the guard of $EPQ_1^>$. Let O_c be the set of operations in e that have priorities comparable with *pri*, and Let O_i be the set of operations in e that have priorities incomparable with *pri*. It is obvious that l_1 is the linearization of $e|_{O_c}$. By Lemma 24, there exists sequence l, such that $e \sqsubseteq l$, and $l|_{O_c} = l_1$. Then $l = u' \cdot put(x,pri) \cdot v' \cdot rm(x) \cdot w'$, where $u'|_{O_c} = u$, $v'|_{O_c} = v$ and $w'|_{O_c} = w$. Since *pri* is one of maximal priorities in e, and the predicates of guards of $EPQ_1^>$ does not restrict O_i, it is easy to see that $l \in MS(EPQ_1^>)$ and then $e \sqsubseteq l$ with witness x. □

We can see that $UVSet_i(e,x) \cap UVSet_j(e,x) = \emptyset$ for any $i \neq j$.

The following lemma states that $UVSet(e,x)$ contains only matched *put* and *rm*.

Lemma 25. *Given a data-differentiated pri_x-execution e with $last(e) = EPQ_1^>$. Let $put(x,pri_x)$ and $rm(x)$ be method events of e with maximal priority. Let G be the graph representing the left-right constraint of $put(x,pri_x)$ and $rm(x)$. Assume that G has no cycle going through x. Then, $UVSet(e,x)$ contains only matched put and rm.*

Proof. We prove this lemma by contradiction. Assume that there exists a value, such that $UVSet(e,x)$ contains only its *put* and does not contain its *rm*. Then we can see that there exists $d_1,\dots,d_j$. Intuitively, $d_1,\dots,d_j$ are elements in $UVSet_1(e,x),\dots,UVSet_i(e,x)$, respectively. $UVSet(e,x)$ contains $put(d_j,\_)$ and does not contain $rm(d_j)$. And each d_i is the reason of $d_{i+1} \in UVSet_{i+1}(e,x)$. Formally, we require that

- For each $1 \leq i \leq j$, method events of d_i belongs to $UVSet_i(e,x)$.
- For each $i \neq j$, $put(d_i,\_), rm(d_i) \in UVSet_i(e,x)$. $put(d_j,\_) \in UVSet_j(e,x)$, and e does not contain $rm(d_j)$.
- An operation of d_1 happens before an operations of x. For each $1 < i \leq j$, an operation of d_i happens an operation of d_{i-1}.
- For each k and *ind*, if $k > ind{+}1$, then no operation of d_k happens before operation of d_{ind}.

According to the definition of $UVSet(e,x)$, it is easy to see that such $d_1,\dots,d_j$ exists. Let us prove the following fact:

***fact*$_1$**: Given $1 \leq i < j$, it can not be the case that $put(d_i,\_)$ and $rm(d_i)$ overlap.

Proof of $fact_1$: We prove $fact_1$ by contradiction. Assume that for some $i \neq j$, $put(d_i,\_)$ and $rm(d_i)$ overlap. Since $put(d_i,\_), rm(d_i) \in UVSet_i(h,x)$, we know that an operation o_i of d_i happens before operation o_{i-1} of d_{i-1}. Moreover, since $put(d_i,\_)$ and $rm(d_i)$ overlap, it is not hard to see that the call action of $put(d_i,\_)$ and the call action of $rm(d_i)$ is before the call action of o_{i-1}. Since method events of d_{i+1} is in $UVSet_{i+1}(e,x)$, we know that an operation o'_{i+1} of d_{i+1} happens before operation o'_i of d_i. Then, it is not hard to see that o'_{i+1} also happens before o_{i-1}, which contradicts that for each $k > ind{+}1$, no operation of d_k happens before operation of d_{ind}.

We already know that an operation of d_1 happens before an operation of x. By *fact*$_1$, we can ensure that *put*$(d_1, \_)$ happens before an operation of x, and then $d_1 \rightarrow x$ in G. For each $1 < i \leq j$, we know that an operation o_i of d_i happens before an operation $o_{i\text{-}1}$ of $d_{i\text{-}1}$. By *fact*$_1$, we can ensure that o_i = *put*$(d_i, \_)$ and $o_{i\text{-}1}$ = *rm*$(d_{i\text{-}1})$, and then $d_i \rightarrow d_{i\text{-}1}$ in G. Since h contains *put*$(d_j, \_)$ and does not contain *rm*(d_j), we know that $x \rightarrow d_j$ in G. Then G has a cycle going through x, contradicts that G has no cycle going through x. □

The following lemma states that *UVSet*(e, x) does not happen before *rm*(x) when the left-right constraint has no cycle going through x.

Lemma 26. *Given a data-differentiated pri*$_x$*-execution* e *with last*(e) = *EPQ*$_1^{>}$*. Let put*(x, pri_x) *and rm*(x) *be method events of* e *with maximal priority. Let* G *be the graph representing the left-right constraint of put*(x, pri_x) *and rm*(x)*. Assume that* G *has no cycle going through* x*. Then, rm*(x) *does not happen before any operation in UVSet*(e, x).

Proof. We prove this lemma by induction, and prove that *rm*(x) does not happen before any operation in *UVSet*$_1(e, x)$, in *UVSet*$_2(e, x)$, ... Note that, by Lemma 25, *UVSet*(e, x) contains only matched *put* and *rm*, and it is easy to see that for each i, *UVSet*$_i(e, x)$ contains only matched *put* and *rm*.
(1) Let us prove that *rm*(x) does not happen before any operation in *UVSet*$_1(e, x)$ by contradiction. Assume that *rm*$(x) <_{hb} o$, where $o \in$ *UVSet*$_1(e, x)$ is an operation of item d.

We use a triple (t_1, t_2, t_3) to represent related information. t_1, t_2, t_3 are chosen from $\{$*put*, *rm*$\}$. t_1 represents whether o is a *put* method event or a *rm* method event. t_2 and t_3 is used for the reason of $o \in$ *UVSet*$_1(e, x)$: $o \in$ *UVSet*$_1(e, x)$, since an operation (of kind t_2) of d happens before an operation (of kind t_3) of x. Let us consider all the possible cases of (t_1, t_2, t_3):

- (*put*, *put*, *put*): Then *rm*$(x) <_{hb}$ *put*$(d, \_) <_{hb}$ *put*(x, pri_x), contradicts that *rm*(x) does not happen before *put*(x, pri_x).
- (*put*, *put*, *rm*): Then *rm*$(x) <_{hb}$ *put*$(d, \_) <_{hb}$ *rm*(x), contradicts that *rm*(x) does not happen before *rm*(x).
- (*put*, *rm*, *put*): Then (*rm*$(x) <_{hb}$ *put*$(d, \_)) \wedge ($*rm*$(d) <_{hb}$ *put*$(x, pri_x))$. By interval order, we know that (*rm*$(x) <_{hb}$ *put*$(x, pri_x)) \vee ($*rm*$(d) <_{hb}$ *put*$(d, \_))$, which is impossible.
- (*put*, *rm*, *rm*): Then (*rm*$(x) <_{hb}$ *put*$(d, \_)) \wedge ($*rm*$(d) <_{hb}$ *rm*$(x))$. We can see that *rm*$(d) <_{hb}$ *rm*$(x) <_{hb}$ *put*$(d, \_)$, which contradicts that *rm*(d) does not happen before *put*$(d, \_)$.
- (*rm*, *put*, *put*): Then (*rm*$(x) <_{hb}$ *rm*$(d)) \wedge ($*put*$(d, \_) <_{hb}$ *put*$(x, pri_x))$. We can see that x and d has circle in G, contradicts that G has no cycle going through x.
- (*rm*, *put*, *rm*): Then (*rm*$(x) <_{hb}$ *rm*$(d)) \wedge ($*put*$(d, \_) <_{hb}$ *rm*$(x))$. We can see that x and d has circle in G, contradicts that G has no cycle going through x.
- (*rm*, *rm*, *put*): Then *rm*$(x) <_{hb}$ *rm*$(d) <_{hb}$ *put*(x, pri_x), contradicts that *rm*(x) does not happen before *put*(x, pri_x).

- (rm, rm, rm): Then $rm(x) <_{hb} rm(d) <_{hb} rm(x)$, contradicts that $rm(x)$ does not happen before $rm(x)$.

This completes the proof for $UVSet_1(e,x)$.

(2) Assume we already prove that for some $j \geq 1$, $rm(x)$ does not happen before any operation in $UVSet_1(e,x) \cup \ldots \cup UVSet_j(e,x)$. Let us prove that $rm(x)$ does not happen before any operation in $UVSet_{j+1}(e,x)$ by contradiction. Assume that $rm(x) <_{hb} o$, where $o \in UVSet_{j+1}(e,x)$ is an operation of item d_{j+1}. We use a triple (t_1, t_2, t_3) to represent related information. t_1, t_2, t_3 are chosen from $\{put, rm\}$. t_1 represents whether o is a *put* method event or a *rm* method event. t_2 and t_3 is used for the reason of $o \in UVSet_{j+1}(e,x)$: $o \in UVSet_{j+1}(e,x)$, since an operation (of kind t_2) of d_{j+1} happens before an operation (of kind t_3) of d_j, where $put(d_j, \_), rm(d_j) \in UVSet_j(e,x)$. Let us consider all the possible cases of (t_1, t_2, t_3):

- (put, put, put): Then $rm(x) <_{hb} put(d_{j+1}, \_) <_{hb} put(d_j, \_)$. We can see that $(rm(x) <_{hb} put(d_j, \_)) \wedge (put(d_j, \_) \in UVSet_j(e,x))$, which contradicts that $rm(x)$ does not happen before any operation in $UVSet_1(e,x) \cup \ldots \cup UVSet_j(e,x)$.
- (put, put, rm): Then $rm(x) <_{hb} put(d_{j+1}, \_) <_{hb} rm(d_j, \_)$. We can see that $(rm(x) <_{hb} rm(d_j, \_)) \wedge (rm(d_j) \in UVSet_j(e,x))$, which contradicts that $rm(x)$ does not happen before any operation in $UVSet_1(e,x) \cup \ldots \cup UVSet_j(e,x)$.
- (put, rm, put): Then $(rm(x) <_{hb} put(d_{j+1}, \_)) \wedge (rm(d_{j+1}) <_{hb} put(d_j, \_))$. By interval order, we know that $(rm(x) <_{hb} put(d_j, \_)) \vee (rm(d_{j+1}) <_{hb} put(d_{j+1}, \_))$, which is impossible.
- (put, rm, rm): Then $(rm(x) <_{hb} put(d_{j+1}, \_)) \wedge (rm(d_{j+1}) <_{hb} rm(d_j))$. By interval order, we know that $(rm(x) <_{hb} rm(d_j)) \vee (rm(d_{j+1}) <_{hb} put(d_{j+1}, \_))$, which is impossible.
- (rm, put, put): Then $(rm(x) <_{hb} rm(d_{j+1})) \wedge (put(d_{j+1}, \_) <_{hb} put(d_j, \_))$. Let us consider the reason of $put(d_j, \_), rm(d_j) \in UVSet_j(e,x)$:
  - If $(j > 1) \wedge (put(d_j, \_) <_{hb} o'')$, where o'' is an operation of item d_{j-1} and $put(d_{j-1}, \_), rm(d_{j-1}) \in UVSet_{j-1}(e,x)$: Then since $(put(d_{j+1}, \_) <_{hb} put(d_j, \_)) \wedge (put(d_j, \_) <_{hb} o'')$, we can see that $put(d_{j+1}, \_) <_{hb} o''$, and then operations of d_{j+1} is in $UVSet_j(e,x)$, contradicts that operations of d_{j+1} is in $UVSet_{j+1}(e,x)$.
  - If $(j = 1) \wedge (put(d_j, \_) <_{hb} o'')$, where o'' is an operation of x: Similar to above case.
  - If $(j > 1) \wedge (rm(d_j) <_{hb} o'')$, where o'' is an operation of item d_{j-1} and $put(d_{j-1}, \_), rm(d_{j-1}) \in UVSet_{j-1}(e,x)$: Then since $(put(d_{j+1}, \_) <_{hb} put(d_j, \_)) \wedge (rm(d_j) <_{hb} o'')$, we can see that $(put(d_{j+1}, \_) <_{hb} o'') \vee (rm(d_j) <_{hb} put(d_j, \_))$, which is impossible.
  - If $(j > 1) \wedge (rm(d_j) <_{hb} o'')$, where o'' is an operation of x: Similar to above case.
- (rm, put, rm): Let T_{ind} be the set of sentences $\{rm(x) <_{hb} rm(d_{j+1}), put(d_{j+1}, \_) <_{hb} rm(d_j), \ldots, put(d_{ind+1}, \_) <_{hb} rm(d_{ind})\}$. Here each d_i is a item of some operation in $UVSet_i(e,x)$. Let us prove that from T_j we can obtain contradiction by induction:
  Base case 1: From T_1 we can obtain contradiction.
  Let us prove base case 1:

- If *put*$(d_1, \_)$ happens o, and o is an operation of x. Then there is a cycle $x \rightarrow d_{j+1} \rightarrow \ldots \rightarrow d_1 \rightarrow x$ in G, contradicts that G has no cycle going through x.
- If *rm*(d_1) happens before o, and o is an operation of x. Then since *put*$(d_2, \_) <_{hb}$ *rm*(d_1) and *rm*$(d_1) <_{hb} o$, we can see that *put*$(d_2, \_) <_{hb} o$, and then *put*$(d_2, \_) \in$ *UVSet*$_1(e, x)$, contradicts that *put*$(d_2, \_) \in$ *UVSet*$_2(e, x)$.

Base case 2: From T_2 we can obtain contradiction.

Let us prove base case 2: If *rm*$(d_2) <_{hb} o$, and o is an operation of d_1, then since $($*put*$(d_3, \_) <_{hb}$ *rm*$(d_2)) \wedge ($*rm*$(d_2) <_{hb} o)$, we know that *put*$(d_3, \_) <_{hb} o$. This implies that *put*$(d_3, \_) \in$ *UVSet*$_2(e, x)$, contradicts that *rm*$(d_3, \_) \in$ *UVSet*$_3(e, x)$. Therefore, it is only possible that *put*$(d_2, \_)$ happens before an operation of d_1.

- If *put*$(d_2, \_) <_{hb}$ *put*$(d_1, \_)$ and *put*$(d_1, \_)$ happens before operations of x, then we know that *put*$(d_2, \_)$ happens before operation of x, which is impossible.
- If *put*$(d_2, \_) <_{hb}$ *put*$(d_1, \_)$ and *rm*(d_1) happens before operations of x, then by interval order, we know that *put*$(d_2, \_)$ happens before operation of x, or *rm*$(d_1) <_{hb}$ *put*$(d_1, \_)$, which is impossible.
- If *put*$(d_2, \_) <_{hb}$ *rm*(d_1) and *put*$(d_1, \_)$ happens before operations of x, then $x \rightarrow d_{j+1} \rightarrow \ldots \rightarrow d_1 \rightarrow x$ in G, contradicts that G has no cycle going through x.
- If *put*$(d_2, \_) <_{hb}$ *rm*(d_1) and *rm*(d_1) happens before operations of x, then we know that *put*$(d_2, \_)$ happens before operation of x, which is impossible.

induction step: Given $ind \geq 3$, if from T_{ind-1} we can obtain contradiction, then from T_{ind} we can also contain contradiction.

Prove of the induction step: Similarly as base case 2, we can prove that it is only possible that *put*$(d_{ind}, \_)$ happens before operations of d_{ind-1}.

- If *put*$(d_{ind}, \_) <_{hb}$ *put*$(d_{ind-1}, \_)$ and *put*$(d_{ind-1}, \_)$ happens before operations of d_{ind-2}, then we know that *put*(d_{ind}) happens before operation of d_{ind-2}, which is impossible.
- If *put*$(d_{ind}, \_) <_{hb}$ *put*$(d_{ind-1}, \_)$ and *rm*(d_{ind-1}) happens before operations of d_{ind-2}, then by interval order, we know that *put*$(d_{ind}, \_)$ happens before operation of d_{ind-2}, or *rm*$(d_{ind-1}) <_{hb}$ *put*$(d_{ind-1}, \_)$, which is impossible.
- If *put*$(d_{ind}, \_) <_{hb}$ *rm*(d_{ind-1}), then we obtain T_{ind-1}, which already contain contradiction.

By base case 1, base case 2 and the induction step, it is easy to see that for each i, T_i contains contradiction. Therefore, T_j, the case of (*rm*, *put*, *rm*), contains contradiction.

- (*rm*, *rm*, *put*): Then $($*rm*$(x) <_{hb}$ *rm*$(d_{j+1})) \wedge ($*rm*$(d_{j+1}) <_{hb}$ *put*$(d_j, \_))$. We can see that $($*rm*$(x) <_{hb}$ *put*$(d_j, \_)) \wedge ($*put*$(d_j, \_) \in$ *UVSet*$_j(e, x))$, which contradicts that *rm*(x) does not happen before any operation in *UVSet*$_1(e, x) \cup \ldots \cup$ *UVSet*$_j(e, x)$.
- (*rm*, *rm*, *rm*): Then $($*rm*$(x) <_{hb}$ *rm*$(d_{j+1})) \wedge ($*rm*$(d_{j+1}) <_{hb}$ *rm*$(d_j))$. We can see that $($*rm*$(x) <_{hb}$ *rm*$(d_j)) \wedge ($*rm*$(d_j) \in$ *UVSet*$_j(e, x))$, which contradicts that *rm*(x) does not happen before any operation in *UVSet*$_1(e, x) \cup \ldots \cup$ *UVSet*$_j(e, x)$.

This completes the proof for *UVSet*$_{j+1}(e, x)$. Therefore, *rm*(x) does not happen before any operation in *UVSet*$(e, x) =$ *UVSet*$_1(e, x) \cup$ *UVSet*$_2(e, x) \cup \ldots$. $\square$

With Lemma 25 and Lemma 26, we can now prove Lemma 9.

Lemma 9. *Given a data-differentiated pri_x-execution e with $last(e) = EPQ_1^>$. Let $put(x, pri_x)$ and $rm(x)$ be method events of e with maximal priority. Let G be the graph representing the left-right constraint of $put(x)$ and $rm(x)$. $e \sqsubseteq MS(EPQ_1^>)$, if and only if G has no cycle going through x.*

Proof. To prove the *only if* direction, assume that $e \sqsubseteq MS(EPQ_1^>)$. Let u, v and w be the sequences of method events in $EPQ_1^>$, and let U, V and W be the set of method events of u, v and w, respectively. Assume by contradiction that, there is a cycle $d_1 \rightarrow d_2 \rightarrow \ldots \rightarrow d_m \rightarrow x \rightarrow d_1$ in G. It is obvious that the priority of each d_i is smaller than pri_x. Then our proof proceeds as follows:

According to the definition of left-right constraint, there are two possibilities. The first possibility is that, $rm(x)$ happens before $rm(d_1)$. It is obvious that $rm(d_1) \in W$, and then since $U \cup V$ contains matched *put* and *rm*, we can see that $put(d_1), rm(d_1) \in W$. Then,

- Since $d_1 \rightarrow d_2$, by definition of G, we know that $put(d_1)$ happens before $rm(d_2)$. Since $put(d_1) \in W$ and $U \cup V$ contains matched *put* and *rm*, we know that $put(d_2), rm(d_2) \in W$. Similarly, for each $1 \leq i \leq m$, we know that $put(d_i), rm(d_i) \in W$.
- Since $d_m \rightarrow x$,
  - if $put(d_m)$ happens before $put(x)$, then we can see that $put(d_m) \in U$, which contradicts that $put(d_m) \in W$.
  - if $put(d_m)$ happens before $rm(x)$, then we can see that $put(d_m) \in U \cup V$, which contradicts that $put(d_m) \in W$.

The second possibility is that, e contains one $put(d_1, \_)$ and no $rm(d_1)$. Note that for each $j > 1$, e contains $put(d_j, \_)$ and $rm(d_j)$. Since $d_m \rightarrow x$, is is obvious that $put(d_m) \in U \cup V$. Since $U \cup V$ contains matched *put* and *rm*, we know that $put(d_m), rm(d_m) \in U \cup V$. Then, since $d_{m\text{-}1} \rightarrow d_m$, by definition of G, we know that $put(d_{m\text{-}1})$ happens before $rm(d_m)$. Since $rm(d_m) \in U \cup V$ and $U \cup V$ contains matched *put* and *rm*, we know that $put(d_{m\text{-}1}), rm(d_{m\text{-}1}) \in U \cup V$. Similarly, for each $1 < i \leq m$, we know that $put(d_i), rm(d_i) \in U \cup V$, and also $put(d_1) \in U \cup V$. However, there is one $put(d_1, \_)$ and no $rm(d_1)$ in e, contradicts that $U \cup V$ contains matched *put* and *rm*.

This completes the proof of the *only if* direction.

To prove the *if* direction, assume that G has no cycle going through x. Let E_u be the set of operations that happen before $put(x)$ in e. It is easy to see that $E_u \subseteq UVSet(e, x)$. Let $E_v = UVSet(e, x) \setminus E_u$. Let E_e be the set of operations of e, and let $E_w = E_e \setminus UVSet(e, x)$.

By Lemma 25, we can see that $E_u \cup E_v$ contains matched *put* and *rm* operations. It remains to prove that for E_u, $\{put(x, pri_x)\}$, E_v, $\{rm(x)\}$, E_w, no elements of the latter set happens before elements of the former set. We prove this by showing that all the following cases are impossible:

- Case 1: Some operation $o_w \in E_w$ happens before $rm(x)$. Then we know that $o_w \in UVSet(e, x) = E_u \cup E_v$, which contradicts that $o_w \in E_w$.
- Case 2: Some operation $o_w \in E_w$ happens before some operation $o_{uv} \in E_u \cup E_v$. Then we know that $o_w \in UVSet(e, x) = E_u \cup E_v$, which contradicts that $o_w \in E_w$.

- Case 3: Some operation $o_w \in E_w$ happens before *put*(x). Then we know that $o_w \in \mathit{UVSet}(e,x) = E_u \cup E_v$, which contradicts that $o_w \in E_w$.
- Case 4: *rm*(x) happens before some $o_{uv} \in \mathit{UVSet}(e,x) = E_u \cup E_v$. By Lemma 26 we know that this is impossible.
- Case 5: *rm*(x) happens before *put*(x). This contradicts that each single-priority projection satisfy the FIFO property.
- Case 6: Some operation $o_v \in E_v$ happens before *put*(x). Then we know that $o_v \in E_u$, which contradicts that $o_v \in E_v$.
- Case 7: Some operation $o_v \in E_v$ happens before some operation $o_u \in E_u$. Then we know that $o_v \in E_u$, which contradicts that $o_v \in E_v$.
- Case 8: *put*(x) happens before some operation $o_u \in E_u$. This is impossible.

This completes the proof of the *if* direction.

□

Let us begin to represent witness automata that is used for capture the existence of a data-differentiated execution e, e has a _-projection e', $\mathit{last}(e') = \mathit{PQ}_1^{>}$, and there exists a cycle going through the item with maximal priority in e'. By data-independence, we can obtain e_r from e by renaming function, which maps such item to be b, maps items that cover it to be a, and maps other items into d. There are four possible enumeration of call and return actions of *put*(b) and *rm*(b). For each of them, we generate a witness automaton.

For the case when $e_r|_b = \mathit{cal}(\mathit{put},b,p) \cdot \mathit{ret}(\mathit{put}) \cdot \mathit{cal}(\mathit{rm}) \cdot \mathit{ret}(\mathit{rm},b)$, we generate witness automaton $\mathcal{A}^1_{\text{l-lar}}$, as shown in Fig. 14. Here $c_1 = c + \mathit{ret}(\mathit{rm},a)$, $c_2 = c + \mathit{cal}(\mathit{put},a,\mathit{les}_p)$, $c_3 = c_2 + \mathit{ret}(\mathit{rm},a)$, where $c = \mathit{cal}(\mathit{put},d,\mathit{anyPri}), \mathit{ret}(\mathit{put},d), \mathit{cal}(\mathit{rm},d), \mathit{ret}(\mathit{rm},d), \mathit{cal}(\mathit{rm},\mathit{empty}), \mathit{ret}(\mathit{rm},\mathit{empty})$. The differentiated branch in $\mathcal{A}^1_{\text{l-lar}}$ comes from the positions of the first *ret*(put,a).

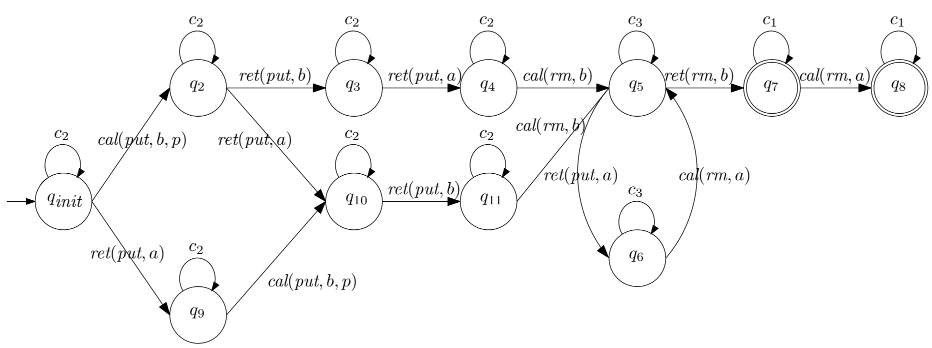

Fig. 14. Automaton $\mathcal{A}^1_{\text{l-lar}}$

$\mathcal{A}^1_{\text{l-lar}}$ is used to recognize conditions in Fig. 15. Here for simplicity, we only draw operation of b, and the first *ret*(put,a).

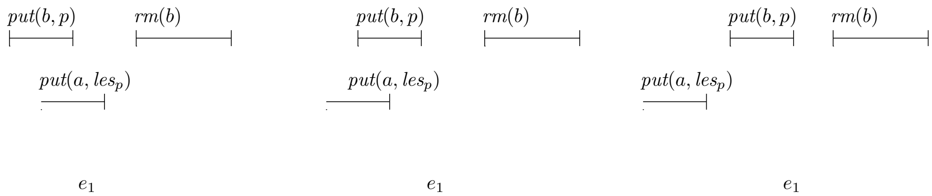

Fig. 15. Conditions recognized by $\mathcal{A}^1_{\textit{l-lar}}$

For the case when $e_r|_b = \textit{cal}(\textit{put}, b, p) \cdot \textit{cal}(\textit{rm}) \cdot \textit{ret}(\textit{put}) \cdot \textit{ret}(\textit{rm}, b)$, we generate witness automaton $\mathcal{A}^2_{\textit{l-lar}}$, as shown in Fig. 16. Here c_1, c_2, c_3 is the same as that in $\mathcal{A}^1_{\textit{l-lar}}$. The differentiated branch in $\mathcal{A}^2_{\textit{l-lar}}$ comes from the positions of the first $\textit{ret}(\textit{put}, a)$.

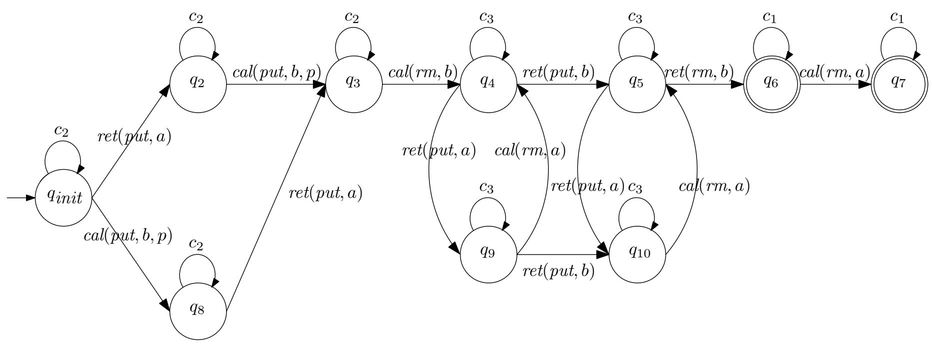

Fig. 16. Automaton $\mathcal{A}^2_{\textit{l-lar}}$

$\mathcal{A}^2_{\textit{l-lar}}$ is used to recognize conditions in Fig. 17. Here for simplicity, we only draw operation of b, and the first $\textit{ret}(\textit{put}, a)$.

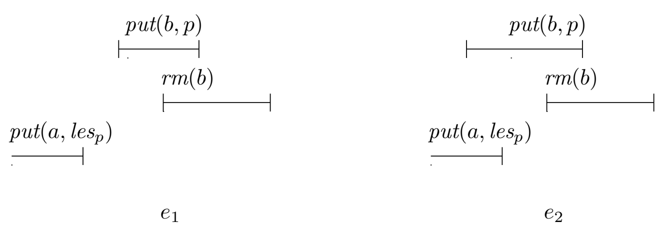

Fig. 17. Conditions recognized by $\mathcal{A}^2_{\textit{l-lar}}$

For the case when $e_r|_b = \textit{cal}(\textit{rm}) \cdot \textit{cal}(\textit{put}, b, p) \cdot \textit{ret}(\textit{put}) \cdot \textit{ret}(\textit{rm}, b)$, we generate witness automaton $\mathcal{A}^3_{\textit{l-lar}}$, as shown in Fig. 18. Here c_1, c_2, c_3 is the same as that in $\mathcal{A}^1_{\textit{l-lar}}$. The differentiated branch in $\mathcal{A}^3_{\textit{l-lar}}$ comes from the positions of the first $\textit{ret}(\textit{put}, a)$.

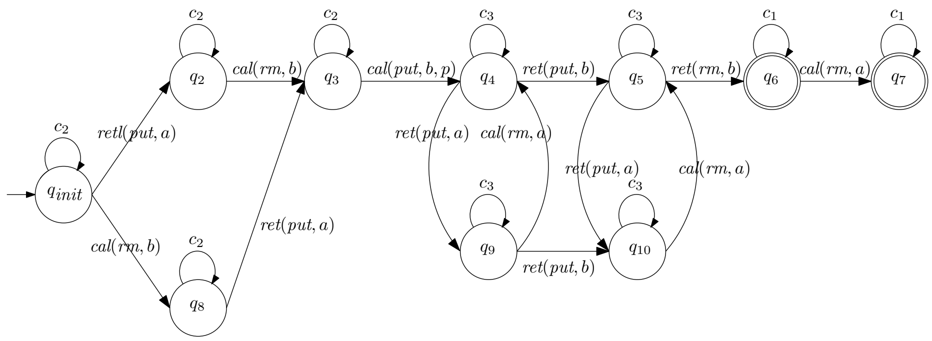

Fig. 18. Automaton $\mathcal{A}^3_{\textit{l-lar}}$

$\mathcal{A}^3_{\textit{l-lar}}$ is used to recognize conditions in Fig. 19. Here for simplicity, we only draw operation of b, and the first *ret*(put, a).

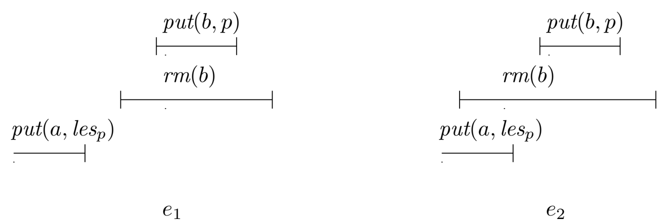

Fig. 19. Conditions recognized by $\mathcal{A}^3_{\textit{l-lar}}$

For the case when $e_r|_b = \textit{cal}(\textit{rm}) \cdot \textit{cal}(\textit{put}, b, p) \cdot \textit{ret}(\textit{rm}, b) \cdot \textit{ret}(\textit{put})$, we generate witness automaton $\mathcal{A}^4_{\textit{l-lar}}$, as shown in Fig. 20. Here c_1, c_2, c_3 is the same as that in $\mathcal{A}^1_{\textit{l-lar}}$, and $c_4 = c_1 + \textit{ret}(\textit{put}, b)$. The differentiated branch in $\mathcal{A}^4_{\textit{l-lar}}$ comes from the positions of the first *ret*(put, a).

$\mathcal{A}^4_{\textit{l-lar}}$ is used to recognize conditions in Fig. 21. Here for simplicity, we only draw operation of b, and the first *ret*(put, a).

Let $\textit{Auts}_{\textit{1-lar}} = \{\mathcal{A}^1_{\textit{l-lar}}, \mathcal{A}^2_{\textit{l-lar}}, \mathcal{A}^3_{\textit{l-lar}}, \mathcal{A}^4_{\textit{l-lar}}\}$. The following lemma states that $\textit{EPQ}^{>}_1$ is co-regular.

Lemma 10. *$\textit{EPQ}^{>}_1$ is co-regular.*

Proof. We need to prove that, given a data-independence implementation $\mathcal{I}$, $\textit{Auts}_{\textit{1-lar}} \cap \mathcal{I} \neq \emptyset$, if and only if $\exists e \in \mathcal{I}_{\neq}$, $e' \in \textit{proj}(e)$, $\textit{EPQ}^{>}_1 \in last(e') \wedge e'$ does not linearizable w.r.t. $\textit{MS}(\textit{EPQ}^{>}_1)$.

By Lemma 8 and Lemma 9, we need to prove the following fact:

fact$_1$: Given a data-independence implementation $\mathcal{I}$, $\textit{Auts}_{\textit{1-lar}} \cap \mathcal{I} \neq \emptyset$ if and only if $\exists e \in \mathcal{I}_{\neq}$, $e' \in \textit{proj}(e)$, $last(e') = \textit{EPQ}^{>}_1$, x is the item with maximal priority *pri* in e', e' is a *pri*-execution. And there is a cycle going through x in G, where G is the left-right constraint of e'.

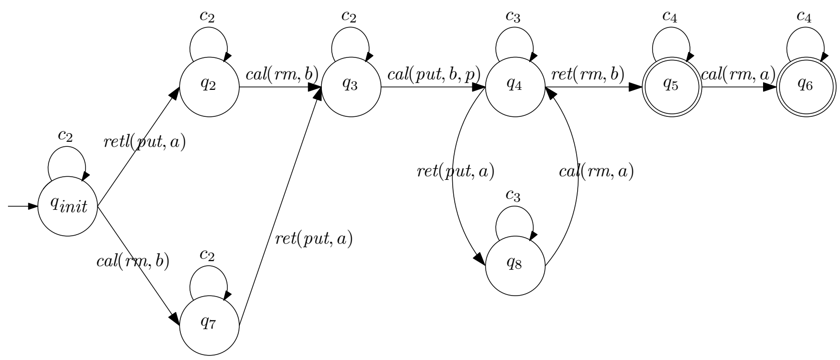

Fig. 20. Automaton $\mathcal{A}^4_{\textit{l-lar}}$

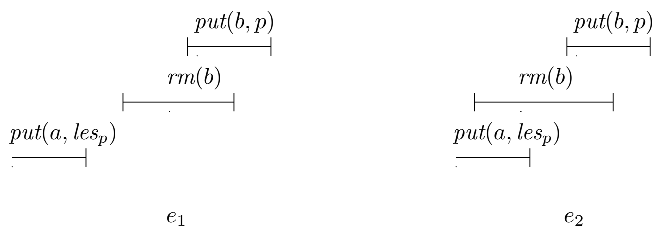

Fig. 21. Conditions recognized by $\mathcal{A}^4_{\textit{l-lar}}$

The *only if* direction: Let us consider the case of $\mathcal{A}^1_{\textit{l-lar}}$. Assume that $e_1 \in \mathcal{I}$ is accepted by $\mathcal{A}^1_{\textit{l-lar}}$. By data-independence, there exists data-differentiated execution $e \in \mathcal{I}$ and renaming function r_1, such that $e_1 = r_1(e)$. Assume that r_1 maps d into b and maps $f_1, \dots, f_m$ into a. Let e' be obtained from e by projection into $\{d, f_1, \dots, f_m\}$. Assume that the priority of b is p. It is easy to see that e' is a p-execution, $\textit{last}(e') = \textit{EPQ}^{>}_1$, and there is a cycle going through d in G, where G is the left-right constraint of e'. The case of $\mathcal{A}^2_{\textit{l-lar}}$, $\mathcal{A}^3_{\textit{l-lar}}$ and $\mathcal{A}^4_{\textit{l-lar}}$ can be similarly proved.
The *if* direction: Given such e, e' and x. Let renaming function r maps x into b, maps items cover x into a, and maps other items into d. By data-independence, $r(e) \in \mathcal{I}$. Then depending on the cases of $r(e)|_b$, we can see that $r(e)$ is accepted by $\mathcal{A}^1_{\textit{l-lar}}$, $\mathcal{A}^2_{\textit{l-lar}}$, $\mathcal{A}^3_{\textit{l-lar}}$ or $\mathcal{A}^4_{\textit{l-lar}}$. $\square$

D.3 Proofs and Definitions in Subsection 6.3

Let $\textit{Items}(e, p)$ be the set of items with priority p in execution e. The following lemma states a method to choose *itm* of $\textit{EPQ}^{=}_1$.

Lemma 27. *Given a data-differentiated pri-execution e with $\textit{last}(e) = \textit{EPQ}^{=}_1$. If there exists an item x with priority pri, such that for each $y \in \textit{Items}(e, pri)$, (1) x does not $<_{pb}$ to y, and (2) the right-most gap-point of x is after $\textit{cal}(put, y, \_)$ and $\textit{cal}(rm, y)$. Then $e \sqsubseteq \textit{MS}(\textit{EPQ}^{=}_1)$.*

Proof. Let o be the right-most gap-point of x. We locate linearization points of each method event as follows:

- Locate the linearization point of *rm*(x) at o,
- If *put*(x, pri) overlaps with *rm*(x), then locate the linearization point of *put*(x, pri) just before the linearization point of *rm*(x). Otherwise, *put*$(x, pri) <_{hb}$ *rm*(x), and we locate the linearization point of *put*(x, pri) just before its return action.
- Locate linearization points of method event of each $y \in$ *Items*(e, pri) (except for x) just after the call action of the method event.
- For item z with priority smaller than *pri*. If both *cal*$(put, z, \_)$ and *cal*(rm, z) is before o, then locate the linearization points of *put*$(z, \_)$ and *rm*(z) just after their call actions. If both *ret*(put, z) and *ret*(rm, z) (if exists) is after o, then locate the linearization points of *put*$(z, \_)$ and *rm*(z) just before their return actions. Otherwise, x is in interval of z, which contradicts the definition of gap-point, and is impossible.

Let l be the sequence of linearization points constructed above. It is obvious that $e \sqsubseteq l$. Since for each $y \in$ *Items*(e, pri), o is after *cal*$(put, y, \_)$ and *cal*(rm, x), we can see that *rm*(x) is after *put*(y, pri) and *rm*(y) in l. It is obvious that *put*(x, pri) is before *rm*(x) in l. Since x does not $<_{pb}$ to y, we can see that no *put*(y, pri) happens before *put*(x, pri). Then it is easy to see that *put*(x, pri) is after *put*(y, pri) in l. Since *last*$(e) = EPQ_1^{=}$, all other items in *Items*(e, pri) has matched *put* and *rm*, and it is easy to see that their *put* and *rm* (except for that of x) are all before *rm*(x) in l.

For item z with priority smaller than *pri*, we can see that there are only two possibilities: (1) *put*$(z, \_)$ and *rm*(z) are both before *rm*(x) in l, and (2) *put*$(z, \_)$ and *rm*(z) (if exists) are after before *rm*(x) in l. Therefore, before *rm*(x) in l, the *put* and *rm* of z are matched.

Therefore, it is easy to see that $l \in MS(EPQ_1^{=})$. □

With Lemma 27, we can prove the following lemma, which states that getting rid of case in Fig. 7 is enough for ensure *last*$(e) = EPQ_1^{=} \Rightarrow e \sqsubseteq MS(EPQ_1^{=})$.

Lemma 11. *Given a data-differentiated pri-execution* e *with last*$(e) = EPQ_1^{=}$. e *does not linearizable w.r.t* $MS(EPQ_1^{=})$*, if and only if there exists* x *and* y *with maximal priority pri in* e*, such that* $y <_{pb}^{*} x$ *in* e*, and the rightmost gap-point of* x *is before cal*(put, y, pri) *or cal*(rm, y) *in* e*.*

Proof. To prove the *if* direction, let $e_{x,y}$ be the execution that is obtained from e by erasing all actions of items that has same priority as x, except for actions of x and y. It is obvious that *last*$(e_{x,y}) = EPQ_1^{=}$. Since $y <_{pb}^{*} x$, according to $EPQ_1^{=}$, we can see that x should be chosen as *itm* in $EPQ_1^{=}$.

According to Lemma 9 (Here we temporarily forget the existence of y), the only possible position for locating linearizaton point of *rm*(x) is at gap-point of x. Otherwise, if the linearizaton point of *rm*(x) is chosen at a position that is not a gap-point of x, then there exists unmatched method event before *rm*(x) with smaller priority. Since the rightmost gap-point of x is before *cal*(put, y, pri) or *cal*(rm, y), if we locate linearizaton point of *rm*(x) at gap-point of x, then *rm*(x) will be before *cal*(put, y, pri) or *cal*(rm, x).

Therefore, for every sequence $l = u \cdot put(x, pri) \cdot v \cdot rm(x) \cdot w$, if $e_{x,y} \sqsubseteq l$, then either $u \cdot v$ contains some unmatched method events of priority smaller than *pri*, or w contains $put(y, pri)$ or $rm(y)$. In both cases, $l \notin MS(EPQ_1^{=})$.

To prove the *only if* direction, we prove its contrapositive. Assume we already know that for each x and y has maximal priority in e, if $y <_{pb}^* x$, then the rightmost gap-point of x is after $cal(put, y, pri)$ and $cal(rm, x)$. We need to prove that $e \sqsubseteq MS(EPQ_1^{=})$. Recall that we already assume that each single-priority execution has FIFO property, and item with larger priority is not covered by items with smaller priority.

Our proof proceed as follows:

- Let e_{pri} be the projection of e into operations of priority *pri*. Since each single-priority execution has FIFO property, there exists sequence l_{pri}, such that $e_{pri} \sqsubseteq l_{pri}$, and when we treat *put* as *enq* and *rm* as *deq*, l_{pri} belongs to queue.
- Let a_1 be the last inserted item of l_{pri}.
  Step 1: Check whether for each $b \in Items(e, pri)$, (1) a_1 does not $<_{pb}$ to b, and (2) the right-most gap-point of a is after $cal(put, b, pri)$ and $cal(rm, b)$.
  It is easy to see that a_1 is of priority *pri*, and a_1 does not $<_{pb}$ to any $b \in Items(e, pri)$. If for each $b \in Items(e, pri)$, the rightmost gap-point of a_1 is after $cal(put, b, pri)$ and $cal(rm, b)$. Then by Lemma 27, we can obtain that $e \sqsubseteq MS(EPQ_1^{=})$.
- Otherwise, there exists $a_2 \in Items(e, pri)$, such that the rightmost gap-point of a_1 is before $cal(put, a_2, pri)$ or $cal(rm, a_2)$ in e. We can see that each gap-point of a_2 is after the rightmost gap-point of a_1.By assumption, we know that a_2 does not $<_{pb}$ to a_1.
  - If for each item $b \in Items(e, pri)$, a_2 does not $<_{pb}$ to b. Then we go to step 1 and treat a_2 similarly as a_1.
  - Otherwise, there exists a_3 with priority *pri* such that $a_2 <_{pb}^* a_3$.
    Since l_{pri} has FIFO property, it is easy to see that there is no cycle in $<_{pb}$ order. It is safe to assume that a_3 is maximal in the sense of $<_{pb}^*$. Or we can say, there does not exists a_4, such that $a_3 <_{pb}^* a_4$.
    By assumption,we know that the rightmost gap-point of a_3 is after $cal(put, a_2, pri)$ and $cal(rm, a_2)$. Therefore, we can see that the rightmost gap-point of a_3 is after the rightmost gap-point of a_1. Then we go to step 1 and treat a_3 similarly as a_1.

Let a^i be the a_1 in the *i-th* loop of our proof. It is not hard to see that, given $i < j$, the rightmost gap-point of a^j is after the rightmost gap-point of a^i. Therefore, the loop finally stop at some a^f. a^f satisfies the check of Step 1. By Lemma 27, this implies that $e \sqsubseteq MS(EPQ_1^{=})$. This completes the proof of *if* direction. $\square$

According to the definition of $<_{ob}^*$, if $a <_{pb}^* b$, then there exists $a_1, \ldots, a_m$, such that $a <_{pb} a_1 <_{pb} \ldots <_{pb} a_m <_{pb} b$. The following lemma states that, the number of intermediate items a_i is in fact bounded.

Lemma 12. *Given a data-differentiated execution h. Assume that $a <_{pb} a_1 <_{pb} \ldots <_{pb} a_m <_{pb} b$, then one of the following cases holds:*

- $a <_{pb}^A b$, $a <_{pb}^B b$ *or* $a <_{pb}^C b$,

- $a <^A_{pb} a_i <^B_{pb} b$, *or* $a <^B_{pb} a_i <^A_{pb} b$, *for some* i.

Proof. Our proof proceed as follows:

- ($<^A_{pb} \cdot <^A_{pb}$,$<^B_{pb} \cdot <^B_{pb}$ and $<^C_{pb} \cdot <^C_{pb}$): If $c_3 <^A_{pb} c_2 <^A_{pb} c_1$, then $put(c_3, \_)$ happens before $put(c_2, \_)$, and $put(c_2, \_)$ happens before $put(c_1, \_)$. Therefore, it is obvious that $put(c_3, \_)$ happens before $put(c_1, \_)$ and $c_3 <^A_{pb} c_1$.
  Similarly, if $c_3 <^B_{pb} c_2 <^B_{pb} c_1$, then $c_3 <^B_{pb} c_1$.
  If $c_3 <^C_{pb} c_2 <^C_{pb} c_1$: Since $c_2 <^C_{pb} c_1$, $ret(rm, c_2)$ is before $cal(put, c_1, \_)$. Since $rm(c_2)$ does not happen before $put(c_2, \_)$, $cal(put, c_2, \_)$ is before $ret(rm, c_2)$. Since $c_3 <^C_{pb} c_2$, $ret(rm, c_3)$ is before $cal(put, c_2, \_)$. Therefore, $ret(rm, c_3)$ is before $cal(put, c_1, \_)$, and $c_3 <^C_{pb} c_1$.
  Therefore, when we meet successive $<^A_{pb}$, it is safe to leave only the first and the last elements and ignore intermediate elements. Similar cases hold for $<^B_{pb}$ and $<^C_{pb}$.
- $<^A_{pb}$ and $<^C_{pb}$:
  - ($<^A_{pb} \cdot <^C_{pb}$): If $c_3 <^A_{pb} c_2 <^C_{pb} c_1$. Since $c_2 <^C_{pb} c_1$, $ret(rm, c_2)$ is before $cal(put, c_1, \_)$. Since $rm(c_2)$ does not happen before $put(c_2, \_)$, $cal(put, c_2, \_)$ is before $ret(rm, c_2)$. Since $c_3 <^A_{pb} c_2$, $ret(put, c_3)$ is before $cal(put, c_2, \_)$. Therefore, $ret(put, c_3)$ is before $cal(put, c_1, \_)$, and $c_3 <^A_{pb} c_1$.
  - ($<^C_{pb} \cdot <^A_{pb}$): If $c_3 <^C_{pb} c_2 <^A_{pb} c_1$. Since $c_2 <^A_{pb} c_1$, $ret(put, c_2)$ is before $cal(put, c_1, \_)$. It is obvious that $cal(put, c_2, \_)$ is before $ret(put, c_2)$. Since $c_3 <^C_{pb} c_2$, $ret(rm, c_3)$ is before $cal(put, c_2, \_)$. Therefore, $ret(rm, c_3)$ is before $cal(put, c_1, \_)$, and $c_3 <^C_{pb} c_1$.
- $<^B_{pb}$ and $<^C_{pb}$:
  - ($<^B_{pb} \cdot <^C_{pb}$): If $c_3 <^B_{pb} c_2 <^C_{pb} c_1$. Since $c_2 <^C_{pb} c_1$, $ret(rm, c_2)$ is before $cal(put, c_1, \_)$. It is obvious that $cal(rm, c_2)$ is before $ret(rm, c_2)$. Since $c_3 <^B_{pb} c_2$, $ret(rm, c_3)$ is before $cal(rm, c_2)$. Therefore, $ret(rm, c_3)$ is before $cal(put, c_1, \_)$, and $c_3 <^C_{pb} c_1$.
  - ($<^C_{pb} \cdot <^B_{pb}$): If $c_3 <^C_{pb} c_2 <^B_{pb} c_1$. Since $c_2 <^B_{pb} c_1$, $ret(rm, c_2)$ is before $cal(rm, c_1)$. Since $rm(c_2)$ does not happen before $put(c_2, \_)$, $cal(put, c_2, \_)$ is before $ret(rm, c_2)$. Since $c_3 <^C_{pb} c_2$, $ret(rm, c_3)$ is before $cal(put, c_2, \_)$. Therefore, $ret(rm, c_3)$ is before $cal(rm, c_1)$, and $c_3 <^B_{pb} c_1$.
- ($<^A_{pb} \cdot <^B_{pb} \cdot <^A_{pb}$): If $c_4 <^A_{pb} c_3 <^B_{pb} c_2 <^A_{pb} c_1$:
  - If $cal(rm, c_2)$ is before $cal(put, c_1, \_)$: Since $c_3 <^B_{pb} c_2$, $ret(rm, c_3)$ is before $cal(rm, c_2)$. Then $ret(rm, c_3)$ is before $cal(put, c_1, \_)$, and $c_3 <^C_{pb} c_1$. This implies that $c_4 <^A_{pb} c_3 <^C_{pb} c_1$. According to the fact for $<^A_{pb} \cdot <^C_{pb}$, we know that $c_4 <^A_{pb} c_1$.
  - If $cal(rm, c_2)$ is after $cal(put, c_1, \_)$: Since $c_2 <^A_{pb} c_1$, $ret(put, c_2, \_)$ is before $cal(put, c_1, \_)$. Since $c_3 <^B_{pb} c_2$, $rm(c_3)$ happens before $rm(c_2)$, and then we know that $put(c_2, \_)$ can not happen before $put(c_3, \_)$. Since $put(c_2, \_)$ does not happen before $put(c_3, \_)$, $cal(put, c_3, \_)$ is before $ret(put, c_2, \_)$. Since $c_4 <^A_{pb} c_3$, $ret(put, c_4)$ is before $cal(put, c_3, \_)$. Therefore, $ret(put, c_4)$ is before $cal(put, c_1, \_)$, and $c_4 <^A_{pb} c_1$.

- $(<_{pb}^{B} \cdot <_{pb}^{A} \cdot <_{pb}^{B})$: If $c_4 <_{pb}^{B} c_3 <_{pb}^{A} c_2 <_{pb}^{B} c_1$: Since $c_2 <_{pb}^{B} c_1$, $ret(rm, c_2)$ is before $cal(rm, c_1)$. Since $c_3 <_{pb}^{A} c_2$, we can see that $put(c_3, \_) <_{hb} put(c_2, \_)$. Since each single-priority execution has FIFO property, we know that $rm(c_2)$ does not happen before $rm(c_3)$, and thus, $cal(rm, c_3)$ is before $ret(rm, c_2)$. Since $c_4 <_{pb}^{B} c_3$, $ret(rm, c_4)$ is before $cal(rm, c_3)$. Therefore, $ret(rm, c_4)$ is before $cal(rm, c_1)$, and $c_4 <_{pb}^{B} c_1$.

Based on above results, given $a <_{pb}^{b_1} a_1 <_{pb} \ldots <_{pb}^{b_m} a_m <_{pb}^{b_{m+1}} b$, where each b_i is in $\{A, B, C\}$, we can merge relations, until we got one of the following facts:

- $a <_{pb}^{A} b$, $a <_{pb}^{B} b$ or $a <_{pb}^{C} b$,
- $a <_{pb}^{A} a_i <_{pb}^{B} b$, or $a <_{pb}^{B} a_i <_{pb}^{A} b$, for some i,

This completes the proof of this lemma. □

There are many enumerations of method events of a, b and a_1 that may makes $a <_{pb}^{*} b$. The following lemma states that with the help of gap-points, the number of potential enumerations can be further reduced into only five.

Lemma 13. *Given a data-differentiated pri-execution e with last(e) = $EPQ_1^{=}$. Let a and b be items with maximal priority pri. Assume that $a <_{pb}^{*} b$, and the rightmost gap-point of b is before cal(put, a, pri) or cal(rm, a). Then, there are five possible enumeration of method events of a, b, a_1 (if exists), where a_1 is the possible intermediate items for obtain $a <_{pb}^{*} b$.*

Proof. Let us prove by consider all the possible reason of $a <_{pb}^{*} b$. According to Lemma 12, we need to consider five reasons: Let o be the right-most gap-point of b.

- Reason 1, $a <_{pb}^{A} b$:
  Since $a <_{pb}^{A} b$, $put(a, pri) <_{hb} put(b, pri)$. Since o is after $cal(put, b, pri)$, and thus, after $cal(put, a, pri)$, we can see that o is before $cal(rm, b)$.
  Since single-priority execution must satisfy the FIFO property, $rm(b)$ does not happen before $rm(a)$, and thus, $cal(rm, a)$ is before $ret(rm, b)$. If $cal(rm, a)$ is before $cal(rm, b)$, then o is also a gap-point of a and contradicts our assumption. So we know that $cal(rm, a)$ is after $cal(rm, b)$. If $ret(rm, a)$ is before $ret(rm, b)$, since we already assume that there exists gap-point of a, this gap-point is also a gap-point of b, and is after o, which contradicts that o is the rightmost gap-point of b. Therefore, $ret(rm, a)$ is after $ret(rm, b)$.
  According to above discussion, there are two possible enumeration of operations of a and b, as shown in Fig. 22 and Fig. 23. Here we explicitly draw the leftmost gap-point of a as o'. Since the position of $ret(put, b)$ does not influence the correctness, we can simply ignore it.
- Reason 2, $a <_{pb}^{B} b$:
  Since $a <_{pb}^{B} b$, $ret(rm, a)$ is before $cal(rm, b)$. Since o is after $cal(rm, b)$, we can see that o is before $cal(put, a, pri)$. This implies that $ret(rm, a)$ is before $cal(put, a, pri)$, and then $rm(a) <_{hb} put(a)$, which is impossible. Therefore, we can safely ignore this reason.

- Reason 3, $a <^C_{pb} b$:
  Since $a <^B_{pb} b$, *ret*(rm, a) is before *cal*(put, b, pri). Since o is after *cal*(put, b), we can see that o is before *cal*(put, a, pri). This implies that *ret*(rm, a) is before *cal*(put, a, pri), and then $rm(a) <_{hb} put(a)$, which is impossible. Therefore, we can safely ignore this reason.
- Reason 4, $a <^A_{pb} a_1 <^B_{pb} b$:
  Since $a_1 <^B_{pb} b$, $rm(a_1) <_{hb} rm(b)$, and *ret*(rm, a_1) is before *cal*(rm, b). Since $rm(a_1)$ does not happen before $put(a_1)$, *cal*(put, a_1, pri) is before *ret*(rm, a_1). Since $a <^A_{pb} a_1$, *ret*(put, a, pri) is before *cal*(put, a_1, pri). Therefore, *ret*(put, a, pri) is before *cal*(rm, b). Since *cal*(rm, b) is before o, we can see that o is before *cal*(rm, a). If *cal*(rm, a) is after *ret*(rm, b), then $e|_{\{a, a_1, b\}}$ violates the FIFO property. Therefore, *cal*(rm, a) is before *ret*(rm, b). Similarly as the case of reason 1, we can see that *ret*(rm, b) is before *ret*(rm, a).
  According to above discussion, there are three possible enumeration of operations of a, a_1 and b, as shown in Fig. 24, Fig. 25 and Fig. 26. Here we explicitly draw the leftmost gap-point of a as o'. Since the position of *ret*(put, a_1, pri) and *cal*(put, a, pri) do not influence the correctness, we can simply ignore it. We also ignore *cal*(put, b, pri) and *ret*(put, b), since the only requirements of them are (1) $rm(b)$ does not happen before $put(b)$ and (2) *cal*(put, b, pri) is before o.
- Reason 5, $a <^B_{pb} a_1 <^A_{pb} b$:
  Since $a_1 <^A_{pb} b$, *ret*(put, a_1) is before *call*(put, b, pri). Since *call*(put, b, pri) is before o, we can see that *ret*(put, a_1) is before o.
  - If o is before *cal*(rm, a): Then o is obviously before *ret*(rm, a). Since $a <^B_{pb} a_1$, *ret*(rm, a) is before *cal*(rm, a_1). Then we can see that, o is before *cal*(rm, a_1), and remember that $a_1 <^A_{pb} b$. Then we can goto the case of reason 1 and treat a_1 as a. Therefore, we can safely ignore this.
  - If o is before *cal*(put, a, pri): Since $rm(a)$ does not happen before $put(a, pri)$, we can see that *cal*(put, a, pri) is before *ret*(rm, a), and then o is before *ret*(rm, a). Then similarly as above case, we can see that o is before *cal*(rm, a_1), and $a_1 <^A_{pb} b$. Then we can goto the case of reason 1 and treat a_1 as a. Therefore, we can safely ignore this.

This completes the proof of this lemma. □

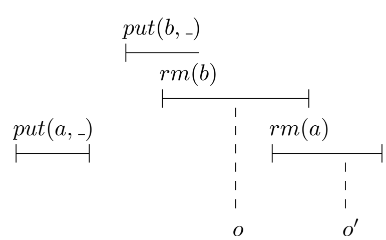

Fig. 22. The first possible enumeration.

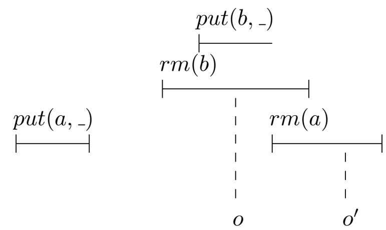

Fig. 23. The second possible enumeration.

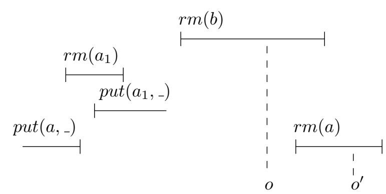

Fig. 24. The third possible enumeration.

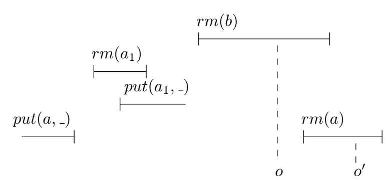

Fig. 25. The forth possible enumeration.

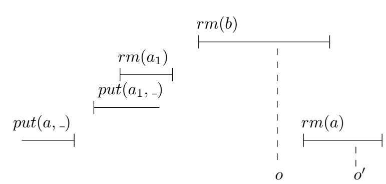

Fig. 26. The fifth possible enumeration.

Let us begin to represent several witness automata that is used to capture the existence of a data-differentiated $\_$-execution e, e has a projection e', $last(e') = EPQ_1^{=}$, there exists items a and b with maximal priority in e', $a <_{pb}^{*} b$, and the rightmost gap-point of b is before $cal(put, a, \_)$ or $cal(rm, a)$.

Given a data-differentiated $\_$-execution e, two actions act_1, act_2 of maximal priority in e, and assume that act_1 is before act_2 in e. we say that act_1, act_2 is covered by items $d_1, \ldots, d_m$ in e, if the priorities of $d_1, \ldots, d_m$ is smaller than that of act_1 and act_2, and

- $ret(put, d_m, \_)$ is before act_1,
- For each $i < 1 \leq m$,$put(d_{i\text{-}1}, \_)$ happens before $rm(d_i)$,
- act_2 is before $cal(rm, d_1)$.

According to Lemma 11, Lemma 12 and Lemma 13, it is not hard to prove that, given a data-differentiated $\_$-execution e with $last(e) = EPQ_1^{=}$, e does not linearizable with respect to $EMS(PQ_1^{=})$, if and only if, one of enumerations holds in e (permit renaming), while $cal(rm, a)$ and $ret(rm, b)$ is covered by some $d_1, \ldots, d_m$, $cal(rm, b)$ is before $ret(put, d_m, \_)$, and $cal(rm, d_1)$ is before $ret(rm, a)$. We say that such $d_1, \ldots, d_m$ constitute the rightmost gap of b.

An automaton $\mathcal{A}_{l\text{-}eq}^1$ is given in Fig. 27, and it is constructed for the first enumeration in Fig. 22. Here we rename the items that covers $cal(rm, a)$ and $ret(rm, b)$ into d, and rename the remanning items into e. In this figure, $c = cal(put, e, anyPri), ret(put, e), cal(rm, e), ret(rm, e), cal(rm, empty), ret(rm, empty)$, $c_1 = c + cal(put, d, les_p)$, $c_2 = c_1 + ret(put, b)$, $c_3 = c_2 + cal(put, d), ret(rm, d)$, $c_4 = c + ret(put, b) + ret(rm, d)$.

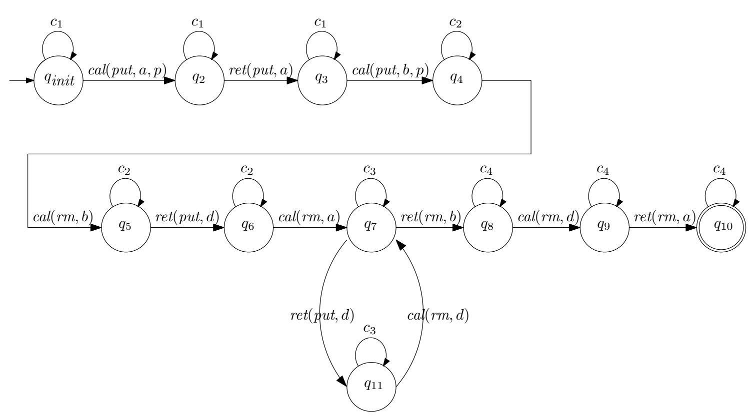

Fig. 27. Automaton $\mathcal{A}_{l\text{-}eq}^1$

An automaton $\mathcal{A}_{l\text{-}eq}^2$ is given in Fig. 28, and it is constructed for the second enumeration in Fig. 23. In Fig. 28, c_1, c_2, c_3 and c_4 is same as that in Fig. 27.

For the third enumeration in Fig. 24. Since we want to ensure that a and b are putted only once, we need to explicitly record the positions of $cal(put, a, p)$ and $cal(put, b, p)$. Since the positions of $cal(put, a, p)$ and $cal(put, b, p)$ are not fixed, there are finite possible cases to consider, as shown below:

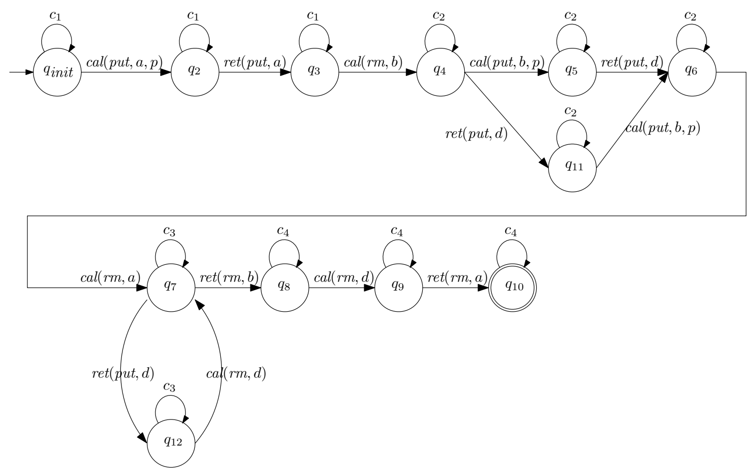

Fig. 28. Automaton $\mathcal{A}^2_{l\text{-}eq}$

- If $cal(put, b, p)$ is after $cal(rm, b)$ and before $cal(rm, a)$: There are two possible positions of $cal(put, a, p)$: (1) before $cal(rm, a_1)$, and (2) after $cal(rm, a_1)$, and before $ret(put, a)$.
- If $cal(put, b, p)$ is after $ret(rm, a_1)$ and before $cal(rm, b)$: same as above case.
- If $cal(put, b, p)$ is after $cal(put, a_1, \_)$ and before $ret(rm, a_1)$: same as above case.
- If $cal(put, b, p)$ is after $ret(put, a)$ and before $cal(put, a_1, p)$: same as above case.
- If $cal(put, b, p)$ is after $cal(rm, a_1)$ and before $ret(put, a)$: There are three possible positions of $cal(put, a, p)$: (1) after $cal(put, b, p)$ and before $ret(put, a)$, (2) after $cal(rm, a_1)$ and before $cal(put, b, p)$, and (3) before $call(rm, a_1)$.
- If $cal(put, b, p)$ is before $cal(rm, a_1)$: There are three possible positions of $cal(put, a, p)$: (1) after $cal(rm, a_1)$ and before $ret(put, a)$, (2) after $cal(put, b, p)$ and before $cal(rm, a_1)$, and (3) before $cal(put, b, p)$.

Therefore, there are fourteen possible cases that satisfy the third enumeration in Fig. 24. For each case, we construct an finite automaton. Let $Auts^3_{1\text{-}eq}$ be the set of finite automata that is constructed for above fourteen cases. For example, for the case ca_1 when $cal(put, a, p)$ is before $cal(rm, a_1)$, $cal(put, b, p)$ is after $ret(rm, a_1)$, and $cal(put, b, p)$ is before $cal(rm, b)$, we construct a finite automaton $\mathcal{A}^{3\text{-}1}_{l\text{-}eq}$ in Fig. 29. In Fig. 29, let c and $c_1 = c + cal(put, d, les_p)$ the same as that in Fig. 27. Let $c_2 = c_1 + ret(put, a_1)$, $c_3 = c_2 + ret(put, b)$, $c_4 = c_3 + cal(put, d) + ret(rm, d)$, and $c_5 = c + ret(put, b) + ret(put, a_1) + ret(rm, d)$. The other witness automata in $Auts^3_{1\text{-}eq}$ can be similarly constructed.

Similarly, we construct sets $Auts^4_{1\text{-}eq}$ and $Auts^5_{1\text{-}eq}$ of witness automata for the forth enumeration in Fig. 25 and the fifth enumeration in Fig. 26, respectively.

Let $Auts_{1\text{-}eq} = \{\mathcal{A}^1_{l\text{-}eq}, \mathcal{A}^2_{l\text{-}eq}\} \cup Auts^3_{1\text{-}eq} \cup Auts^4_{1\text{-}eq} \cup Auts^5_{1\text{-}eq}$. The following lemma states that $PQ^{=}_1$ is co-regular.

Lemma 14. *$EPQ^{=}_1$ is co-regular.*

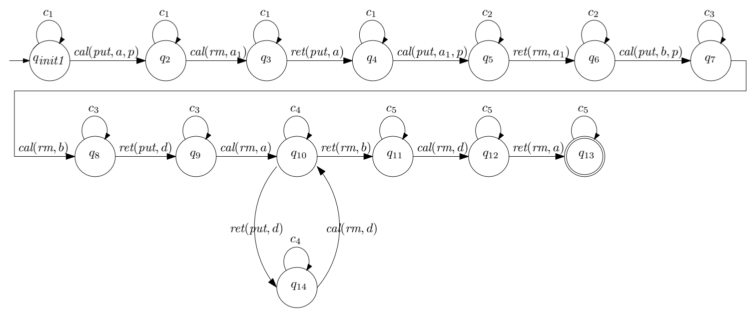

Fig. 29. Automaton $\mathcal{A}^{3\text{-}1}_{l\text{-}eq}$

Proof. We need to prove that, given a data-independence implementation $\mathcal{I}$, $Auts_{1\text{-}eq} \cap \mathcal{I} \neq \emptyset$, if and only if $\exists e \in \mathcal{I}_{\neq}$, $e' \in proj(e)$, $EPQ^{=}_{1} \in last(e') \wedge e'$ does not linearizable w.r.t. $MS(EPQ^{=}_{1})$.

By Lemma 8 and Lemma 11, we need to prove the following fact:

fact$_1$: Given a data-independence implementation $\mathcal{I}$, $Auts_{1\text{-}eq} \cap \mathcal{I} \neq \emptyset$ if and only if $\exists e \in \mathcal{I}_{\neq}$, $e' \in proj(e)$, $last(e') = EPQ^{=}_{1}$, a and b are two items with maximal priority *pri* in e', e' is a *pri*-execution, $a <^{*}_{pb} b$ in e', and the rightmost gap-point of b is before $cal(put, a, pri)$ or $cal(rm, a)$ in e'.

The *only if* direction: Assume that $e_1 \in \mathcal{I}$ is accepted by some witness automata in $Auts_{1\text{-}eq}$. By data-independence, there exists data-differentiated execution $e_2 \in \mathcal{I}$ and a renaming function r, such that $e_1 = r(e_2)$. Since e_1 is accepted by some witness automata in $Auts_{1\text{-}eq}$, let x, y and z (if exists) be the items that are renamed into b, a and a_1 (if exists) by r, respectively, and let $d_1, \ldots, d_m$ be the items that are renamed into d by r.

let $e'' = e_2|_{\{x,y,z,d_1,\ldots,d_m\}}$. It is obvious that $e'' \in proj(e_2)$ is a *pri*-execution, $last(e'') = EPQ^{=}_{1}$. According to our construction of automata in $Auts_{1\text{-}eq}$, it is not hard to see that x and y has maximal priority in h_2, $y <^{*}_{pb} x$, and the rightmost gap-point of x is before $cal(put, y, pri)$ or $cal(rm, y)$ in e''.

The *if* direction: Assume that there exists $e \in \mathcal{I}_{\neq}, e' \in proj(e)$, such that $last(e') = PQ^{=}_{1}$, a' and b' are two items with maximal priority *pri* in e', e' is a *pri*-execution, $a' <^{*}_{pb} b'$ in e', and the rightmost gap-point of b' is before $cal(put, a', pri)$ or $cal(rm, a')$ in e'. By data-independence, we can obtain execution e_1 as follows: (1) rename a' and b' into a and b, respectively, (2) for the items $d_1, \ldots, d_m$ that constitute the rightmost gap of b', we rename them into d, (3) if $a' <^{A}_{pb} a'_1 <^{B}_{pb} b$, we rename a'_1 into a_1, and (4) rename the other items into e. It is easy to see that $last(e_1) = EPQ^{=}_{1}$, a and b has maximal priority in e_1, $a <^{*}_{pb} b$ in e_1, and the rightmost gap-point of b is before $cal(put, a, pri)$ or $cal(rm, a)$ in e_1. By Lemma 13, there are five possible enumeration of operations of a, b, a_1 (if exists). Then

- If $a <^{*}_{pb} b$ because of the first enumeration, it is easy to see that h_1 is accepted by $\mathcal{A}^{1}_{l\text{-}eq}$.
- If $a <^{*}_{pb} b$ because of the second enumeration, it is easy to see that h_1 is accepted by $\mathcal{A}^{2}_{l\text{-}eq}$.

- If $a <^*_{pb} b$ because of the third enumeration, it is easy to see that h_1 is accepted by some witness automaton in $\mathit{Auts}^3_{\mathit{1\text{-}eq}}$.
- If $a <^*_{pb} b$ because of the forth enumeration, it is easy to see that h_1 is accepted by some witness automaton in $\mathit{Auts}^4_{\mathit{1\text{-}eq}}$.
- If $a <^*_{pb} b$ because of the fifth enumeration, it is easy to see that h_1 is accepted by some witness automaton in $\mathit{Auts}^5_{\mathit{1\text{-}eq}}$.

This completes the proof of this lemma. $\square$

D.4 Co-Regular of $\boldsymbol{EPQ_2^{>}}$

Lemma 28. *Given a data-differentiated _-execution e, if $last(e) = EPQ_2^{>}$, then $e \sqsubseteq MS(EPQ_2^{>})$.*

Proof. Since $last(e) = EPQ_2^{>}$, the actions with maximal priority in e is some unmatched *put*. Therefore, no matter how we locate linearization points, we can always obtain a sequence l of method events that contains unmatched *put* with maximal priority, and this satisfy the guard of $MS(EPQ_2^{>})$. This completes the proof of this lemma. $\square$

D.5 Co-Regular of $\boldsymbol{EPQ_2^{=}}$

Lemma 29. *Given a data-differentiated pri-execution e with $last(e) = EPQ_2^{=}$. e does not linearizable to $MS(EPQ_2^{=})$, if and only if there exists x and y with priority pri, x has unmatched put, y has matched put and rm, and $put(x, pri) <_{hb} put(y, pri)$.*

Proof. The *if* direction is obvious.

To prove the *only if* direction, we prove its contrapositive. Assume that for each pair of x and y with maximal priority in e, if x has unmatched *put*, y has matched *put* and *rm*, then $put(x, pri)$ does not happen before $put(y, pri)$. We need to prove that $e \sqsubseteq MS(EPQ_2^{=})$.

Let $x_1, \ldots, x_m$ be the set of items with priority *pri* and has unmatched *put* in e, let $y_1, \ldots, y_n$ be the set of items with priority *pri* and has matched *put* and *rm* in e. By assumption, we know that $cal(put, y_i, pri)$ is before $ret(put, x_j)$ for each i, j. Then we explicitly construction the linearization of e by locating the linearization points of e as follows:

- For each x_i, locate the linearization point of $put(x_i, pri)$ just before its return action.
- For each y_j, locate the lineariztion point of $put(y_j, pri)$ jest after its call action.
- For other method events, locate their linearization points at an arbitrary location after its call action and before its return action.

Let l be the sequence of linearization points. It is easy to see that $e \sqsubseteq l$. Since linearization points of $put(x_i, pri)$ is after the linearization point of $put(y_j, pri)$ for each i, j, it is easy to see that $l \in MS(EPQ_2^{=})$. This completes the proof of this lemma. $\square$

Lemma 29 shows how to check violation to $MS(EPQ_2^{=})$. However, the case in Lemma 29 violates our assumption that each single-priority execution is FIFO. Therefore, we know that $EPQ_2^{=}$ is always co-regular, as states by the following lemma.

Lemma 30. *Given a data-differentiated pri-execution e, if $last(e) = EPQ_2^{=}$, then $e \sqsubseteq MS(EPQ_2^{=})$.*

Proof. According to Lemma 29, if $last(e) = EPQ_2^{=}$ and e does not linearizable to $MS(EPQ_2^{=})$, then there exists x and y with priority *pri*, x has unmatched *put*, y has matched *put*, and $put(x, pri) <_{hb} put(y, pri)$. Let $e_1 = e|_{\{x,y\}}$. It is obvious that e_1 does not satisfy FIFO property. This contradicts the assumption that every single-priority execution has FIFO property, and thus, we can safely ignore this case. $\square$

D.6 Co-Regular of EPQ_3

In this subsection we prove that EPQ_3 is co-regular. The notion of left-right constraint of $rm(empty)$ is inspired by left-right constraint of queue [1].

Definition 12. *Given a data-differentiated execution e, and $o = rm(empty)$ of e. The left-right constraint of o is the graph G where:*

- *the nodes are the items of e or o, to which we add a node,*
- *there is an edge from item d_1 to o, if $put(d_1, \_)$ happens before o,*
- *there is an edge from o to item d_1, if o happens before $rm(d_1)$ or $rm(d_1)$ does not exists in h,*
- *there is an edge from item d_1 to item d_2, if $put(d_1, \_)$ happens before $rm(d_2, \_)$.*

Given a data-differentiated execution e and $o = rm(empty)$ of e, it is obvious that $last(e) = EPQ_3$. Let $USet_1(e, o) = \{op |$ op is an operation of some item, and either $op <_{hb} o$, or there is op' with the same item of op, such that $op' <_{hb} o\}$. For each $i \geq 1$, let $USet_{i+1}(e, o) = \{op |$ op is an operation of some item, op is not in $USet_k(e, o)$ for each $k \leq i$, and either op happens before some $o' \in USet_i(e, o)$, or there is op'' with the same item of o and op'' happens before some $o' \in USet_i(e, o)\}$. We can see that $USet_i(e, o) \cap USet_j(e, o) = \emptyset$ for any $i \neq j$. Let $USet(e, o) = USet_1(e, o) \cup USet_2(e, o) \cup \ldots$.

Similarly as *UVSet*, we can prove the following two lemmas for *USet*.

Lemma 31. *Given a data-differentiated execution e with $last(e) = EPQ_3$. Let o be a $rm(empty)$ of e. Let G be the graph representing the left-right constraint of o. Assume that G has no cycle going through o. Then, $USet(e, o)$ contains only matched put and rm.*

This Lemma can be similarly proved as Lemma 25.

Lemma 32. *Given a data-differentiated _execution e with $last(e) = EPQ_3$. Let o be a $rm(empty)$ of e. Let G be the graph representing the left-right constraint of o. Assume that G has no cycle going through o. Then, o does not happen before any operation in $USet(e, o)$.*

This Lemma can be similarly proved as Lemma 26.

Then we can prove that getting rid of cycle though o in left-right constraint is enough for ensure linearizable w.r.t $MS(EPQ_3)$, as stated by the following lemma.

Lemma 33. *Given a data-differentiated execution e with $last(e) = EPQ_3$. e does not linearizable w.r.t $MS(PQ_3)$, if and only if there exists $o = rm(empty)$ in e, G has a cycle going through o, where G is the graph representing the left-right constraint of o.*

Proof. To prove the *if* direction, assume that there is such a cycle. Assume by contradiction that $e \sqsubseteq MS(EPQ_3)$, and let U and V be the set of operations in u and v. Let the cycle be $d_1 \rightarrow d_2 \rightarrow \ldots \rightarrow d_m \rightarrow o \rightarrow d_1$ in G. Since $d_m \rightarrow o$, $put(d_m, \_)$ happens before o, and it is easy to see that $put(d_m, \_)$ is in U. Since U contains matched *put* and *rm*, we can see that operations of d_m is in U. Similarly, we can see that method events of $d_{m\text{-}1}, \ldots, d_1$ is in U. If $rm(d_1)$ does not exists, then this contradicts that U contains matched *put* and *rm*. Else, if $rm(d_1)$ exists, since o happens before $rm(d_1)$, we can see that $rm(d_1) \in V$, which contradicts that $rm(d_1) \in U$. This completes the proof of the *if* direction.

To prove the *only if* direction, we prove its contrapositive. Assume that for each such o and G, G has no cycle going through o. Let O be the set of operations of e, except for $rm(empty)$. Let $O_L = USet(e, o)$, $O_R = O \setminus O_L$.

By Lemma 31, we can see that $O_L = USet(e, o)$ contains only matched *put* and *rm*. Let O'_L be the union of O_L and all the $rm(empty)$ that happens before some operations in $O_L \cup \{o\}$. Let O'_R be the union of O_R and the remanning $rm(empty)$. It remains to prove that for O'_L, $\{o\}$, O'_R, no elements of the latter set happens before elements of the former set. We prove this by showing that all the following cases are impossible:

- Case 1: If some operation $o_r \in O'_R$ happens before o. Then we can see that $o_r \in USet(e, o)$ or is a $rm(empty)$ that happens before o, and then $o_r \in O'_L$, which contradicts that $o_r \in O'_R$.
- Case 2: If some operation $o_r \in O'_R$ happens before some operation $o_l \in O'_L$. Then we know that $o_r \in USet(e, o)$ or is a $rm(empty)$ that happens before some operations in $O_L \cup \{o\}$, and then $o_r \in O'_L$, which contradicts that $o_r \in O'_R$.
- Case 3: If o happens before some $o_l \in O'_L$. If $o_l \in USet(e, o)$, then by Lemma 32 we know that this is impossible. Else, o_l is a $rm(empty)$ that happens before some operations in $O_L \cup \{o\}$, and o happens before some operations in $O_L \cup \{o\}$, which is impossible by Lemma 32.

This completes the proof of the *only if* direction.

□

Let us begin to represent an automaton that is used for capture the case that, in a sub-execution e' of an execution e, $last(e') = EPQ_3$, e' does not linearizable to $MS(EPQ_3)$, and the reason is that there is a cycle going through some $rm(empty)$ o in the left-right constraint of o. The automaton is $\mathcal{A}^3_{EPQ}$, which is given in Fig. 30. In Fig. 30, let $c = cal(put, d, anyPri), ret(put, d), cal(rm, d), ret(rm, d), cal(rm, empty), ret(rm, empty)$, $c_1 = c + cal(put, b, anyPri)$, $c_2 = c_1 + ret(rm, b)$, and $c_3 = c + ret(rm, b)$.

Given a data-differentiated execution e, we say that $o = rm(empty)$ in e is covered by items $d_1, \ldots, d_m$ in h, if

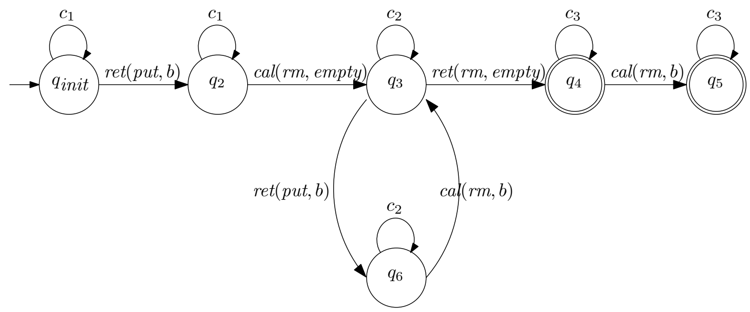

Fig. 30. Automaton $\mathcal{A}^3_{EPQ}$

- $put(d_m, \_)$ happens before o,
- For each $i < 1 \leq m$,$put(d_{i\text{-}1}, \_)$ happens before $rm(d_i)$,
- o happens before $rm(d_1)$, or $rm(d_1)$ does not exists in e

According to the definition of left-right constraint for o, in a data-differentiated execution e, there is a cycle going through o, if and only if there exists items $d_1, \ldots, d_m$, such that o is covered by $d_1, \ldots, d_m$.

Lemma 34. *EPQ_3 is co-regular.*

Proof. We need to prove that, given a data-independence implementation $\mathcal{I}$, $\mathcal{A}^3_{EPQ} \cap \mathcal{I} \neq \emptyset$ if and only if $\exists e \in \mathcal{I}_{\neq}, e' \in proj(e), last(e') = EPQ_3 \wedge e$ does not linearizable w.r.t. $MS(EPQ_3)$.

By Lemma 33, we need to prove the following fact:

fact$_1$: Given a data-independence implementation $\mathcal{I}$, $\mathcal{A}^3_{EPQ} \cap \mathcal{I} \neq \emptyset$ if and only if $\exists e \in \mathcal{I}_{\neq}, e' \in proj(e), last(e') = EPQ_3$, $o = rm(empty)$ is in e', and o is covered by some items $d_1, \ldots, d_m$ in e'.

The *only if* direction: Assume that $e_1 \in \mathcal{I}$ is accepted by $\mathcal{A}^3_{EPQ}$. By data-independence, there exists data-differentiated execution $e_2 \in \mathcal{I}$ and a renaming function r, such that $e_1 = r(e_2)$. Let $d_1, \ldots, d_m$ be the items in e_2 such that $r(d_i) = b$ for each $1 \leq i \leq m$. Let $e_3 = e_2|_{\{o,d_1,\ldots,d_m\}}$. It is obvious that $e_3 \in proj(e_2)$ and $last(e_3) = EPQ_3$. It is easy to see that o is covered by $d_1, \ldots, d_m$.

The *if* direction: Assume that there exists such e, e', o and $d_1, \ldots, d_m$. Then, let e_1 be obtained from e by renaming $d_1, \ldots, d_m$ into b and renaming other items into d. By data-independence, $e_1 \in \mathcal{I}$. It is easy to see that e_1 is accepted by $\mathcal{A}^3_{EPQ}$.

This completes the proof of this lemma. □