Reachability Questions on Partially Lossy Queue Automata

Chris Köcher

Technische Universität Ilmenau, Fachgebiet Automaten und Logik
chris.koecher@tu-ilmenau.de

Data structures are possibly the most important concept in computer science. Famous data structures are, e.g., finite memories, counters, and Turing-tapes. But the most fundamental ones are stacks and queues. Although both data structures have the same set of operations (reading and writing of a letter \(a\)), there is a big difference from computability’s point of view: if we equip finite automata with a stack (these are the well-known pushdown automata) then these models compute exactly the context-free languages. But if we equip a finite automaton with a queue (these are so-called queue automata) we obtain a Turing-complete computation model [4]. This very strong model can be weakened by allowing the queue to forget any part of its content at any time. We call these queues (fully) lossy queues.

When studying the similarities and differences between these two types of queues we found it convenient to join them into one common model. These are the so-called partially lossy queues (plqs for short). This type of queues allows to specify which letters in the queue can be forgotten at any time and which ones are unforgettable. Some algebraic results on partially lossy queues can be found in [10, 9]. In both papers we studied the behavior of partially lossy queues by considering their transformation monoid.

Here, we discuss some reachability questions concerning automata equipped with partially lossy queues, which we call partially lossy queue automata. To this end, for a set of transformations \(T\) and a language \(L\) of queue contents we define the set \(\text{post}_T(L)\) of all queue contents after execution of a transformation in \(T\) on a partially lossy queue with content in \(L\). In other words, \(L\) represents a sets of inputs into the partially lossy queue automaton, \(T\) the set of all paths in its control component (this is an NFA labeled with atomic operations), and \(\text{post}_T(L)\) the corresponding set of outputs. Then the considered computational problem is defined as follows:

**Problem 1** Given a regular language \(T\) of transformations and a regular language \(L\) of queue contents, compute \(\text{post}_T(L)\).

Partially lossy queue automata with at least one non-forgettable letter are Turing-complete. Hence, \(\text{post}_T(L)\) for these partially lossy queues can be any recursively enumerable language. In this case the language \(\text{post}_T(L)\) can be undecidable even if the language of transformations \(T\) equals \(\{w_1, \ldots, w_n\}^*\) for some transformations \(w_1, \ldots, w_n\).

For fully lossy queues, \(\text{post}_T(L)\) is a regular language since it is downwards closed under the subword ordering [7]. Though, it is not possible to compute a finite automaton accepting \(\text{post}_T(L)\) [11]. But we can construct a Turing machine accepting this language [2, 5]. This is
in some sense optimal since Schnoebelen and Chambart proved in [12, 6] that deciding membership of $\text{post}_T(L)$ is not primitive recursive.

In this paper we consider some restrictions on the language of transformations $T$. On the one hand, we regard regular languages that are closed under certain commutations of the atomic operations which guarantee the same behavior of the queue. In this case for arbitrary partially lossy queues $\text{post}_T(L)$ is effectively regular. Thereby, an NFA accepting this language can be computed in polynomial time. If we use an on-the-fly construction of this automaton we can decide its membership problem using non-deterministic logarithmic space, only. Additionally, if the language $L$ of queue contents is context-free then $\text{post}_T(L)$ is context-free as well.

On the other hand, we consider transformation languages of the form $T^*$ where $T$ is a finite, so-called read-write independent set of transformations. These are sets such that for each two words $s, t \in T$ there is a word $u \in T$ where $u$ and $s$ have the same subsequence of write actions and $u$ and $t$ have the same subsequence of read actions. In this case, we can compute an NFA accepting $\text{post}_{T^*}(L)$ using polynomial space, only. This result also covers the special case $w^*$ for some transformation $w$ which was first studied in [3, 1] for fully reliable and fully lossy queues, respectively. An NFA accepting $\text{post}_{w^*}(L)$ can be computed in polynomial time and, again, its membership problem is in NL.

We may also consider backwards reachability in partially lossy queue automata: for a given language $T$ of transformations and a language $L$ of queue contents, the set $\text{pre}_T(L)$ is the set of all queue contents which can reach contents in $L$ after execution of a transformation in $T$ on a partially lossy queue. Then we regard the following computational problem:

**Problem 2** Given a regular language $T$ of transformations and a regular language $L$ of queue contents, compute $\text{pre}_T(L)$.

While this problem is the same as the one above for fully reliable queues due to self-duality (cf. [8]), there are some differences for arbitrary partially lossy queues. In this general case for fully lossy queues it is possible to compute an NFA accepting $\text{pre}_T(L)$. Though, the complexity for computing this NFA is, again, not primitive recursive.

For transformation languages $T$ that are closed under the special commutations we mentioned above, we can compute an NFA accepting $\text{pre}_T(L)$ using polynomial space. Similarly, for $T^*$ where $T$ is finite, read-write independent we can also compute an NFA accepting $\text{pre}_{T^*}(L)$ using polynomial space. In particular, for $w^*$ with a transformation $w$ this is possible in polynomial time, again.

**References**


