

Binary trees are constant-time catenable queues

ANDREAS ABEL

University of Gothenburg and Chalmers University of Technology, Sweden

KOEN CLAESSEN

University of Gothenburg and Chalmers University of Technology, Sweden

CHRISTIAN SATTTLER

University of Gothenburg and Chalmers University of Technology, Sweden

Abstract

Binary trees are constant-time catenable queues. The paper is backed by a mechanisation in Agda.

1 Introduction

Binary trees are constant-time catenable queues.

This paper is accompanied by an Agda formalization. Checkmarks ✓ are clickable and lead to the Agda formalization of the concept adjacent to the checkmark. The formalization may contain extra detail that we chose to omit in the text to keep the presentation concise.

2 Explicit rotation

We consider a modified API that makes tree rotation a primitive operation, since this simplifies the amortization proofs. Since the contents of the queue does not matter for complexity considerations, only its *structure*, we drop the labels from the tree definition and only consider the cases of an (empty) leaf ε and a binary node constructor $_ \cdot _ \checkmark$. Tree rotation $_r$ is a partial function defined by $((t_1 \cdot t_2) \cdot t_3)r = t_1 \cdot (t_2 \cdot t_3) \checkmark$. Dropping from the front of the queue is represented by the partial function “tail” $_t$ defined by $(\varepsilon \cdot t)t = t \checkmark$. The idea is that the tail of a queue can only be taken if its tree representation has been maximally rotated at the root and thus is of the form $\varepsilon \cdot t$.

2.1 The tree game

To study the amortized complexity, we need to talk about arbitrary sequences of tree operations. We conceptualize such sequences as moves in a game where the arena is a multiset of trees which we call a *forest* ✓. Each move takes out some trees from the forest, applies the corresponding operation on them and puts the result back into the forest ✓. For simplicity we represent the forest as a non-empty list so that we can identify particular trees by their index in the list. Note that taking out trees from the forest and putting a tree back (which we do at the front of the list) typically changes indices of tree in the forest. The moves $m \checkmark$ we are considering are:

$C_{i,j}$	concatenate trees i and j
R_i	rotate tree i , if possible
T_i	take tail of tree i , if possible



© 2027 Copyright held by the owner/author(s).

This work is licensed under a Creative Commons Attribution 4.0 International License.

We write $ts\ m = ts'$ ✓ if move m can be applied to forest ts , yielding forest ts' . For instance, $(t_0, t_1, t_2, t_3)C_{1,2} = (t_1 \cdot t_3, t_0, t_2)$ since after taking out t_1 the second index, 2, refers to t_3 in the remaining forest (t_0, t_2, t_3) . We may omit indices if they are 0 ✓. E.g. $(\varepsilon, \varepsilon, \varepsilon)CC = ((\varepsilon \cdot \varepsilon) \cdot \varepsilon)$ and $(\varepsilon, \varepsilon, \varepsilon)CCRTT = (\varepsilon)$.

It is clear that the only terminal forest is (ε) ; no move is applicable here, but any other forest has at least one move. Any tree that is not ε can either be rotated or truncated to its tail, and if the forest has more than one tree, concatenation is possible. A forest of n leaf-trees ε is called an *initial* forest and written ε^n . We are most interested in move sequences starting in initial forests.

To establish amortized complexity we are interested in the longest move sequences starting from a given forest, and the amount of C, T and R moves in those sequences. It is obvious that the number of C moves in a longest sequence is the number of trees in the forest minus one, since each such move reduces the number of trees by one. Since each T move removes one leaf but the last one, the number of T moves is at most the number of leaves in the forest minus one. The maximal number of R moves is the quantity we are most interested in. If it is linearly bounded by the size of the forest, we can execute rotations for free, yielding amortized constant time for queue extraction which in the general case consists of a number of rotations and then a tail operation. We will prove that in move sequences starting in an initial forest ε^{n+1} the number of rotations will be at most $4n$.

2.2 Resourced execution

To amortize the rotations, we raise some fees k_C and k_T that have to be paid for concatenation and tail operations. Those are stored on a virtual bank account k (attached to a forest as subscript ts_k) so that we can pay a unit cost of 1 for each rotation. For uniformity, let us set $k_R = -1$ and refine move execution by the bank account ✓:

$$ts_k\ m = ts'_{k'} \text{ if } k' = k + k_m \geq 0 \text{ and } ts\ m = ts'.$$

The goal is to show that there are constants k_C and k_T such that for each legal move sequence $ts\ ms = ts'$ there is some initial resource k and some leftover resource $k' \geq 0$ such that $ts_k\ ms = ts'_{k'}$ ✓. Moreover, for an initial forest $ts = \varepsilon^n$ the resourced execution should succeed without initial resources ✓, so $k = 0$.

One solution is $k_C = k_T = 2$, and we prove its correctness via the potential method. To this end, we define the following potential functions on forests ✓ and trees ✓:

$$\begin{aligned} \Phi(t_0, \dots, t_n) &= \sum_{i=0..n} \Phi(t_i) \\ \Phi(\varepsilon) &= \Phi_l(\varepsilon) = \Phi_r(\varepsilon) = 0 \\ \Phi(t_1 \cdot t_2) &= \max(\Phi_l(t_1) + 1, \Phi_r(t_2) - 1) \\ \Phi_l(t_1 \cdot t_2) &= \max(\Phi_l(t_1) + 1, \Phi_r(t_2)) \\ \Phi_r(t_1 \cdot t_2) &= \max(\Phi_r(t_1) + 1, \Phi_r(t_2) - 1) \end{aligned}$$

Clearly, all of these functions yield non-negative integers. The various potential functions on trees are related as follows:

LEMMA 2.1.

- (1) $\Phi_r(t) \leq \Phi_l(t) \leq 1 + \Phi_r(t)$. ✓ ✓
- (2) $\Phi_r(t) \leq \Phi(t) \leq 1 + \Phi_r(t)$. ✓ ✓
- (3) $\Phi_l(t) \leq 1 + \Phi(t)$. ✓

PROOF. Easy arithmetical proofs by induction on t . $\square$

THEOREM 2.2. *If $tsms = ts'$ and $k \geq \Phi(ts)$ then $ts_k ms = ts'_{k'}$ for some $k' \geq \Phi(ts')$.* $\checkmark$

PROOF. By induction on the moves $\checkmark$. It is sufficient to consider single moves.

Case $m = C_{i,j}$. Then there is some partition of ts into two trees t_1 and t_2 and some rest ts' . We have to show that for $k \geq \Phi(t_1) + \Phi(t_2) + \Phi(ts')$ there is some $k' \geq \Phi(t_1 \cdot t_2) + \Phi(ts')$ such that $k + 2 = k'$. It is sufficient to show that $\Phi(t_1) + \Phi(t_2) + 2 \geq \Phi(t_1 \cdot t_2) = \max(\Phi_l(t_1) + 1, \Phi_r(t_2) - 1)$ $\checkmark$. This follows from $\Phi(t_1) + 2 \geq \Phi_l(t_1) + 1$ and $\Phi(t_2) + 2 \geq \Phi_r(t_2) - 1$, both implied by Lemma 2.1 $\checkmark$.

Case $m = T_i$. Then forest ts is partitioned into a tree $t_i = \varepsilon \cdot t$ and a remainder ts' . If is sufficient to show that $\Phi(\varepsilon \cdot t) + 2 = \max(1, \Phi_r(t)) + 2 \geq \Phi(t)$ $\checkmark$. This follows from Lemma 2.1, $\Phi_r(t) + 1 \geq \Phi(t)$.

Case $m = R_i$. Then forest ts is partitioned into a tree $(t_1 \cdot t_2) \cdot t_3$ and a remainder ts' . It is sufficient to show that $\Phi((t_1 \cdot t_2) \cdot t_3) \geq \Phi(t_1 \cdot (t_2 \cdot t_3)) + 1$ $\checkmark$ but both sides are equal to $\max(\Phi_l(t_1) + 2, \Phi_r(t_2) + 1, \Phi_r(t_3) - 1)$. $\square$

COROLLARY 2.3. *If $\varepsilon^n ms = ts'$ then $\varepsilon^n_0 ms = ts'_{k'}$ for some $k' \geq 0$.* $\checkmark$

2.3 Optimality

We have shown that $k_C = k_T = 2$ is sufficient to amortize all rotations, but is this budget optimal? In the following we show that $k_C + k_T \geq 4$ is necessary to cover arbitrary move sequences from an initial forest. To this end, we look for longest move sequences from ε^{n+1} and check whether the amount of rotations grows towards $4n$.

First, we notice that the move sequence $s_n = C(CR)^{n+1}TR^n$ $\checkmark$ with a R/C ratio of $\frac{2n+1}{n+2}$ $\checkmark$ run on an initial forest produces a list-shaped tree $(\varepsilon \cdot (\varepsilon \cdot (\varepsilon \cdots)))$ with $n + 1$ leaves $\checkmark$. The sequence $l_n = CR(CRTR)^nCRTRTR^n$ $\checkmark$ started at this list reproduces it $\checkmark$ and thus can be repeated, in each iteration producing a R/C ratio of $\frac{4n+3}{n+2}$ $\checkmark$. Finally the sequence $e_n = CR(TR)^nTT$ $\checkmark$ unravels the list into a single leaf $\checkmark$, contributing a R/C ratio of $\frac{n+1}{1}$ $\checkmark$. The sequence $a_n^m = s_n(l_n)^m e_n$ $\checkmark$ gets an accumulated ratio of $\checkmark$

$$\frac{(2n+1) + (4n+3)m + (n+1)}{(n+2) + (n+2)m + 1} = \frac{4nm + 3m + 3n + 2}{nm + 2m + n + 3} = 4 - \frac{5m + n + 10}{nm + 2m + n + 3} = 4 - \frac{p}{q}$$

where $p = 5m + n + 10$ and $q = nm + 2m + n + 3$. Intuitively, this approaches 4 as m and n go towards infinity, since $p/q > 0$ becomes arbitrary small. Formally, we prove that for each $N > 0$ we can pick n and m such that $p/q \leq 1/N$. Indeed $n = m = 6N$ fulfills $pN \leq q$ $\checkmark$ since $pN = (5 \cdot 6N + 6N + 10)N = 36N^2 + 10N < 36N^2 + 12N + 6N + 3 = q$.

2.4 Method

To find move sequences that would allow to prove the lower bound, a [breadth-first solver](#) for the tree game was developed in Haskell using a coding agent. It could only find the longest move sequences up to 15 concatenations before running out of memory after exploring 100 million positions. Yet the [sequences it produced](#) only used concatenation on the right, so a new solver for this restricted “single-tree” game was coded. With still an exponentially growing search space, [it did not get much further](#), exploring 54 million positions for a maximum of 16 concatenations. However, a certain pattern in the longest move sequences was emerging, and, fixing the initial moves, one could see sequences up to 23 concatenations, confirming the trend, however, without the absolute guarantee that those were longest.

To study the obtained move sequences, an [interactive visualisation](#) of the tree game was developed using a coding agent. This allowed to extract the parametric regular move sequence that has been used in Section 2.3 to show the lower bound of 4 on the combined budget of the concatenation and tail operations. Formal Agda proofs that this sequence is possible for all parameter values, that it yields the calculate ratio between rotation and concatenation moves, and that this ration comes arbitrarily close to 4 have then been developed with the assistance of a coding agent.

3 Conclusion

References