Time Complexity of Tape Reduction for Reversible Turing Machines

Holger Bock Axelsen

DIKU, Department of Computer Science, University of Copenhagen
funkstar@diku.dk

Abstract. Studies of reversible Turing machines (RTMs) often differ in their use of *static resources* such as the number of tapes, symbols and internal states. However, the interplay between such resources and computational complexity is not well-established for RTMs. In particular, many foundational results in reversible computing theory are about multitape machines with two or more tapes, but it is non-obvious what these results imply for reversible complexity theory.

Here, we study how the time complexity of multitape RTMs behaves under reductions to one and two tapes. For deterministic Turing machines, it is known that the reduction from k tapes to 1 tape in general leads to a quadratic increase in time. For k to 2 tapes, a celebrated result shows that the time overhead can be reduced to a logarithmic factor. We show that identical results hold for multitape RTMs.

This establishes that the structure of reversible time complexity classes mirrors that of irreversible complexity theory, with a similar hierarchy.

1 Introduction

Turing machines are very robust with respect to variations in static resources: adding extra symbols, tracks, tapes, or the like does not add any functions or languages to the computable set. While there is no impact on computability, the effect on *computational complexity* can be profound. In a series of classic papers, Hartmanis, Hennie, and Stearns [4, 5, 6] established the effect on time complexity of tape reduction for deterministic Turing machines (DTMs). Tape reduction strictly diminishes the class of problems that we can solve within a given (asymptotic) time bound, which shows the existence of a time hierarchy for DTMs. On the other hand, there are also many complexity classes (such as P) which are themselves robust under tape reduction.

It is known that the reversible Turing machines (RTMs) are equally robust computability-wise wrt the number of symbols, tracks and tapes [2, 11]. However, the analogous complexity results are *not* well-established for RTMs. This limits the generality of statements regarding *reversible complexity*. Complexity classes are (usually) defined for 1-tape machines, but many foundational results in reversible computing theory use *multitape* machines. As a familiar example, the Landauer embedding [8], which is integral to Bennett's method [3], yields a

A. De Vos and R. Wille (Eds.): RC 2011, LNCS 7165, pp. 1–13, 2012.

2-tape RTM that recognizes the same language as a given 1-tape TM. Despite this, it is not at all obvious what the 2-tape machine can tell us about the reversible complexities of this language, since this requires a 1-tape machine. Thus, knowing the complexity-wise effects of varying static resources is both useful and necessary for developing complexity theory for reversible Turing machines.[1]

Here, we examine the effect on *time complexity* of *tape reduction* for RTMs (Section 2). By adapting well-known irreversible tape reductions to work reversibly, we show that tape reductions can be as time-efficient for RTMs as they are for DTMs. We give two main results: first, k-to-1 tape reduction (Section 3) can be done at quadratic cost in running time, in that a $t(n)$ time computation with k tapes can be performed in $\mathcal{O}(t(n)^2)$ time with 1 tape. Like with DTMs we show that this is optimal: some computations are $\Omega(n^2)$ time for 1-tape RTMs, but $\mathcal{O}(n)$ time for 2-tape RTMs, so the quadratic increase in simulation time cannot be improved in general. Second, k-to-2 tape reduction (Section 4) can be done with only a log factor overhead, in that a $t(n)$ time computation with a k-tape machine can be performed in $\mathcal{O}(t(n) \log t(n))$ steps with 2 tapes.

2 Reversible Turing Machines

In this paper we consider multitape reversible Turing machines (RTMs). We assume familiarity with RTMs, and shall only briefly describe them here. We refer the reader to [2, 3, 11] for more complete expositions.

A Turing machine is *reversible* iff it is forward and backward deterministic. This intuitively means that at most one rule from the machine's transition function can be applied to any given configuration (forward determinism), and at most one rule leads to any given configuration (backward determinism).

We use a triple format for the rules of the k-tape RTMs in this paper, with two distinct kinds of rule. A *symbol rule* $(q, (s, t), p)$ says that in state q, if the tape heads read symbols $s \in \Sigma^k$ (where Σ is a finite alphabet), write symbols $t \in \Sigma^k$ and change into state p. A *move rule* (q, d, p) says that in state q, move the k tape heads in the directions given by $d \in \{\leftarrow, \downarrow, \rightarrow\}^k$ and change into state p. A machine is *backward deterministic* iff for any distinct pair of rules (q_1, a_1, p_1) and (q_2, a_2, p_2), if $p_1 = p_2$ then $a_1 = (s_1, t_1)$, $a_2 = (s_2, t_2)$, and $t_1 \neq t_2$.

We are mostly concerned with RTMs for decision problems, and in particular their time complexity. We define the *reversible time complexity class* $\mathsf{RevTIME}_k(t(n))$ to be the set of all languages that are decidable[2] by an $\mathcal{O}(t(n))$ time RTM with k work tapes, where n is the length of the input given on one of the work tapes. This is completely analogous to the definition of $\mathsf{DTIME}_k(t(n))$ for deterministic (but not necessarily reversible) TMs. Note that we trivially have $\mathsf{RevTIME}_k(t(n)) \subseteq \mathsf{DTIME}_k(t(n))$ for any number of tapes k and time bound $t(\cdot)$. For single-tape machines ($k = 1$) we shall omit the subscript.

[1] This is still the case if we instead define time complexity by multitape machines.

[2] *Decider* machines halt in an accepting state '*yes*' for all strings in some language; and halt in a reject state '*no*' for all others. *Recognizer* machines (sometimes also called *acceptors* or *semi-deciders*) may diverge on strings not in the language.

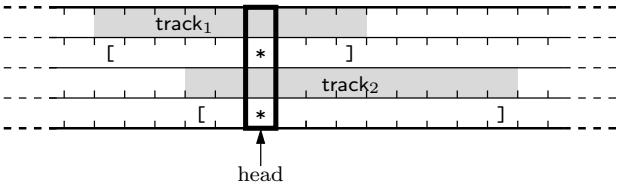
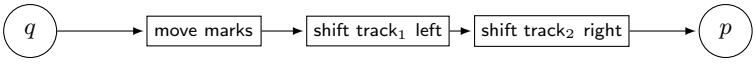


Fig. 1. Track layout for 1-tape simulation of 2-tape Turing machines. Two tracks (track_1 and track_2) simulate the 2 tapes. On auxiliary tracks, delimiters [and] give the extent of the non-blank tape contents (grey squares), and markers $*$ show the simulated tape head positions. After realigning the tracks, these markers always match up.

3 One-Tape Simulation of Multitape RTMs

Hartmanis and Stearns [4] showed that a k-tape Turing machine T_k with running time $t(n)$ can be simulated by a 1-tape TM T_1 with running time $\mathcal{O}(t(n)^2)$. The idea of the simulation is to turn the k tapes into $\mathcal{O}(k)$ *tracks*, with delimiters for the 'ends' of the tapes and markers for the simulated tape heads, as seen in Figure 1. If a multitape rule has the tape heads move differently, *e.g.* $(q, \left[\begin{smallmatrix}\rightarrow\\ \leftarrow\end{smallmatrix}\right], p)$, the simulated rule realigns the tracks such that the markers ($*$) line up again:



The slowdown comes from the fact that shifting the tracks one cell to the left or right takes time linear in their size, and this is done $\mathcal{O}(t(n))$ times. Recently, we applied this idea to RTMs [2]. Importantly, the overhead induced by reversibility does not influence the asymptotic complexities, in that we can still shift the tracks in linear time.

Proposition 1. *A k-tape reversible Turing machine with running time $t(n)$ can be simulated by a 1-tape reversible Turing with running time $\mathcal{O}(t(n)^2)$.*

This establishes an upper bound for general simulation of multitape RTMs.

For DTMs Hennie [5] provided a lower bound matching the upper bound for 1-tape simulation of just 2 tapes. Hennie considers the language

$$L = \{w2^n w \in \{0,1,2\}^* \mid w \in \{0,1\}^n, n > 0\}\,,$$

consisting of strings that repeat a binary word w twice, separated by a block of 2's as long as w itself. The first part of the proof shows a 1-tape TM that decides L in $\mathcal{O}(n^2)$ time, where n is the length of the input. The second part proves that the language L requires at least quadratic time for 1-tape TMs[3] on the 'yes' instances (so the $\mathcal{O}(n^2)$ upper bound is tight). The third part of

[3] In Hennie's and our TM model the input is given on the single work tape. If the input is instead given on a separate read-only (left-to-right) input tape, then the lower bound still holds but is considerably harder to obtain [10].

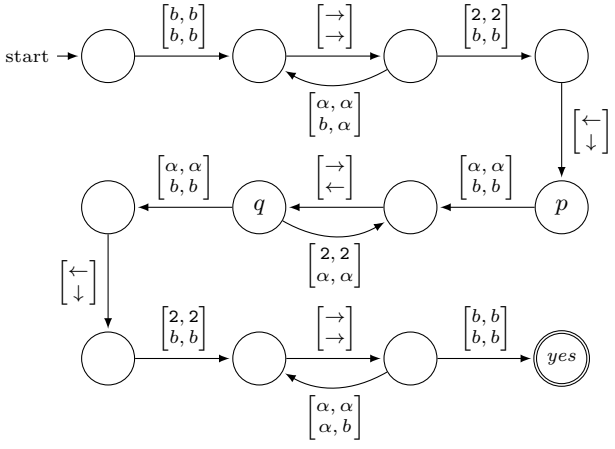


Fig. 2. State transition diagram for a 2-tape RTM that recognizes the language L. Edges with the symbolic variable α are used to signify the two rules for $\alpha \in \{0, 1\}$, and b is the blank symbol.

the proof shows that the language L is decidable by a 2-tape TM in linear time (which is also asymptotically optimal). This shows that the simulation by Hartmanis and Stearns in optimal in general.

This is also the case for RTMs, but in order to prove it we have to replicate each part of Hennie's proof for RTMs. First, we remark that the lower bounds (linear time for 2-tape machines, quadratic time for 1-tape machines) trivially apply to RTMs as well, because the RTMs are a proper subset of the DTMs. However, note that this is not in itself sufficient to show the desired result, because we have not yet established that these lower bounds are *tight* for RTMs. Second, if we can find a 2-tape RTM that decides L in linear time, then we get the quadratic time 1-tape machine for free, simply by applying the 1-tape simulation of 2-tape RTMs. In fact, because the bounds apply specifically to the accepted inputs, a recognizer, rather than a decider, is sufficient. Thus, the entire proof reduces to finding such a machine.

Proposition 2. *There exists a 2-tape RTM that recognizes L in time $\mathcal{O}(n)$.*

Proof. Let the string $l \in \{0, 1, 2\}^*$ be given on the first tape in standard configuration form, *i.e.*, with the tape head to the immediate left of l, and the rest of the tape(s) blank. To decide whether l is in L we follow Hennie's idea: copy the binary word w to the second tape; rewind the second tape while passing over the 2's on the first tape; compare the binary word on the first tape with the copy on the second. Of course, the machine now has to do this reversibly, which complicates matters somewhat, but the basic idea holds. Figure 2 shows an RTM that recognizes L in linear time. □

We can therefore conclude that 1-tape RTM simulation of multitape RTMs has exactly the same time complexity bounds as 1-tape DTM simulation of

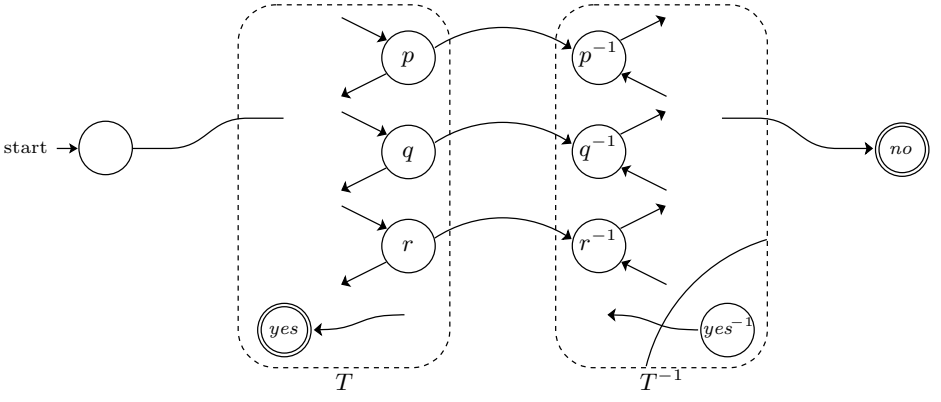


Fig. 3. Composition of T and T^{-1} to eliminate stuck states. The arcs from T to T^{-1} denote the extra symbol rules we add to avoid getting stuck. Note that the inverse image of T's accept state is unreachable.

multitape DTMs. This opens the door for a host of useful and immediate results for reversible complexity theory. For instance, since we have multi-tape universal RTMs that simulate 1-tape machines with constant factor overheads [2,1], standard techniques can be used to show that the reversible Turing machines have a time hierarchy, in a manner completely analogous to DTMs [4]. Furthermore, the Landauer embedding (generalized to k-tape machines) can now be used to show that $\bigcup_k \mathsf{DTIME}_k(t(n)) \subseteq \mathsf{RevTIME}(t(n)^2)$. As a final example, just as P is robust under tape reduction, so is its reversible analogue RevP (defined in the obvious manner).

3.1 Eliminating Stuck States from RTMs

The L-recognizer above gets stuck on all strings that are not in L, *e.g.* at states p and q, rather than rejecting or diverging (running forever). This means that there are two kinds of non-halting behavior: infinite loops and stuck states. To avoid ambiguity it is very often required that machines will *never* get stuck.

It is only possible to get stuck in a state q in some TM T if q does not have outgoing symbol rules for all symbols in the tape alphabet. For irreversible DTMs we can avoid stuck states by simply adding dummy transitions to the reject state for every such undefined transition. For reversible machines this will not work: there is only one reject state, and backward determinism limits the number of transitions into any state to the size of the alphabet. There may be many more stuck states than we can differentiate in this way. A naïve solution could be to expand the (internal) alphabet to include enough symbols to differentiate all the reject transitions we need to add, but this is a somewhat inelegant and wasteful use of resources.

A key feature of RTMs is that they are very easily inverted, cf. [3,2]. Let T^{-1} be the inverse of T, but with the convention that all states q are renamed to

q^{-1} in T^{-1}. We now compose T and T^{-1} such that a stuck state in T will lead to rejection for the composed machine. For each undefined symbol transition for each state q in T we add dummy transitions to state q^{-1} in T^{-1}. For example, if we can get stuck in state p by reading symbol a, then we insert the transition $(p, (a, a), p^{-1})$. Because an outgoing action for this symbol was undefined for p, adding this ingoing edge to p^{-1} will conserve reversibility. If we cross such an added transition, the effect will be to rewind the entire computation leading up to it, leaving us in the state in T^{-1} corresponding to T's starting state. We use this state as the reject state '*no*' of the composed machine.[4] Figure 3 illustrates the construction.

A very useful side effect is that rejection now restores the input string exactly. In fact, by linking T's accept state to a second copy of T^{-1}, we can guarantee that the input string is *always* restored when halting.

In this way it can be ensured that RTMs never get stuck, so we can assume wlog that the only non-halting behavior is running forever. An interesting consequence is that we can then also assume that space-bounded RTMs always halt. Also note that the construction preserves the complexities of the original machine. As a final remark, the author used essentially this construction to implement string comparison in a universal RTM [1].

4 Two-Tape Simulation of Multitape RTMs

Hennie and Stearns [6] showed the beautiful result that a k-tape machine T_k with running time $t(n)$ can be simulated by a 2-tape Turing machine T_2 with only a logarithmic factor slowdown, so that T_2 has running time $\mathcal{O}(t(n) \log t(n))$. Does a similar result hold for the RTMs?

A logarithmic factor is indeed achievable for multitape RTM simulation with only a fixed number of simulation tapes. We can instrument a 2-tape Hennie-Stearns simulation of a k-tape RTM with a Landauer embedding: this yields a 3-tape RTM simulation of the k-tape RTM with only logarithmic factor slowdown.[5]

Thus, only the time complexity of 2-tape reversible simulation of multitape RTMs remains undecided. The rest of this paper is devoted to resolving this problem, by adapting the Hennie-Stearns simulation for reversibility.

4.1 The Hennie-Stearns Simulation

We outline the ideas of the Hennie-Stearns simulation below. For exact details and proofs we refer to the original paper [6], which is still eminently readable.

Layout. The tapes of T_k are each simulated with two tracks placed on one of T_2's tapes (the other tape is for scratch space). A simulated tape is 'horizontally' divided into two *levels* (the two tracks), and 'vertically' into *areas* numbered

[4] If T already has a reject state a dummy move transition targeting its image state in T^{-1} can be added.

[5] It is possible to remove the added trace by Bennett's method for injective functions.

$1, 2, 3, 4, \ldots$ of $1, 2, 4, 8, \ldots$ cells each, on both sides of a designated *home column* H (where the simulated heads of T_k point to). Thus, the ith area on either side has space for 2^i symbols in both levels combined.

The contents of a simulated tape consists of each non-empty area in sequence, first lower levels then upper levels (ignoring empty area levels) reading from the left towards the home column. Symmetrically, from the home column towards the right we read upper levels first, followed by lower levels, for each area. Figure 4(a) shows how to read a simulated tape under this layout.

There are a number of invariants to respect for each simulated tape.

- Each level of each storage area is either completely full or empty.
- The lower level of the home column (the *home square*) is always full, and the upper level always empty.
- If the upper level of an area is full, then so is the lower level *and* the 'mirror' area on the other side of the home column must be empty.
- If the lower level of an area is empty, then so is the upper level *and* the 'mirror' area on the other side of the home column must be completely full.
- At the beginning of the simulation, all lower levels are full, and all upper levels are empty.[6]

This guarantees that the ith right (left) storage area always has space to accommodate the entire non-empty content of the lower-numbered areas to the right (left) of the home square. (Note that a lower level can be full and the upper level empty. Also note that we can have filled lower levels in both right and left area i without breaking the invariants.)

Cleanup. To simulate a move rule, the central idea is to shift only *parts* of the simulated tapes around by using the upper levels as extra storage. For the rest of the paper, assume that we want to simulate rule $(q, [\leftarrow, \cdots], p)$, *i.e.*, that a tape head moves left. We must then shift the simulated tape to the right, which is done as follows.

1. From the home column, find the first non-empty left area, number i.

2a. If the upper level of this area is full, move the upper level's content into the empty lower levels of the $i - 1$ first left areas, with the rightmost symbol going into the home square.

2b. Else, move the lower level of area i in the same way.

3. Collect the original home square symbol and the contents in both levels of the first $i - 1$ *right* areas (these are all full), and place the first half of the content in the lower levels of the first $i - 1$ right areas.

4a. If the lower level of the ith right area is empty, place the second half there.

4b. Else, place the remaining symbols in the (empty) upper level of this area.

5. Return to the home column, and continue with the next simulated tape.

This procedure is called a *cleanup of order* i, and can be executed by the 2-tape DTM in time $\mathcal{O}(2^i)$, *i.e.*, *linearly* in the number of moved symbols. This is key

[6] This means that blank squares from T_k are *not* the same as empty squares in T_2.

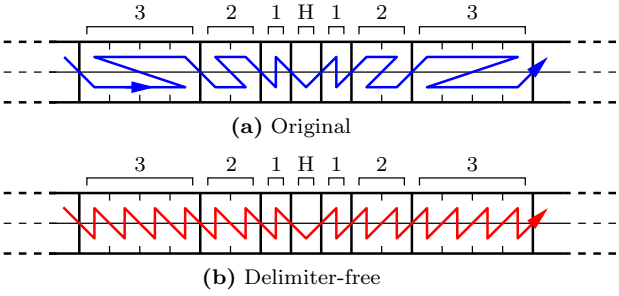


Fig. 4. Layout of a tape simulated with two tracks. The storage areas are numbered around the home column H. (a) shows how to read off the contents of the tape in the original layout by Hennie and Stearns, and (b) shows our modified layout. In both layouts empty squares should be ignored in the reading.

to proving the time bound: the procedure guarantees that high-order cleanups will be very rare compared to low-order cleanups, and this leads to a logarithmic (rather than linear) time factor overhead for the simulation.

4.2 Reversible Cleanup Implementation

Now, can we implement the cleanup procedure reversibly?

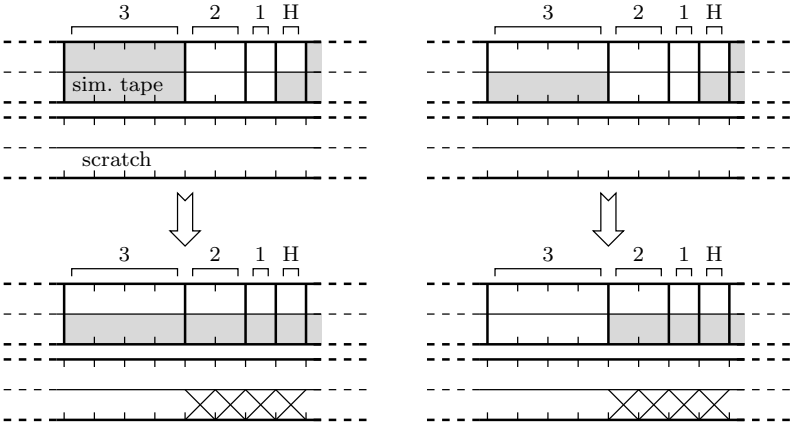
The key idea of the implementation is to use the scratch tape for several purposes. To find the ith left area, we move the simulation tape head to the left until we encounter a non-empty cell: this cell is the right end of area i. At the same time, we also move the scratch tape head to mark out 2^{i-1} cells. This is exactly the length of area i, and also exactly as many symbols as we need to move into the lower order areas. We can also recognize the left end of area i by moving the scratch head back across the markings, while moving the simulation tape head to the left. Thus, the marked scratch space can be used to move exactly the number of symbols we need, in each step of the procedure.

Moving n contiguous symbols n places can easily be done in linear time by RTMs with 2 tapes. It is not difficult to see how to adapt this to the particular kinds of moves we need, and we assume these for the rest of the implementation.

We have identified three main sources of irreversibility in the cleanup procedure, all of which can be dealt with while preserving the time complexity.

Block delimiters. In step 3, we need delimiters to mark the extent of *each* area we visit, in order to move the content from the simulation tape onto the scratch tape in the correct order. In the original implementation delimiters are placed at run-time, but we cannot do this reversibly without knowing by other means whether a delimiter should be placed or not.

Solution. Delimiters can be avoided by changing the tape layout slightly. We adopt a uniform *down/up* reading for each *cell*, rather than for each *area level* (symmetrically, *up/down* to the right of the home column), as shown in Figure 4(b). It is now straightforward to move the home square and right areas onto the scratch tape in the correct order *without* using delimiters between areas.



(2a) upper level full (2b) upper level empty

Fig. 5. Resolving intermediate control flow confluence in step 2 of a 3rd order cleanup. Grey squares are filled, white squares are empty. The two branches are orthogonal by whether area 3 (identified by markings on the scratch tape) is empty or not.

Intermediate control confluence. There is control flow confluence in step 2, when moving the upper level (2a) or lower level (2b) of the ith left area. The control flow of these two branches must be merged reversibly before step 3, which requires finding an orthogonal property to distinguish them by. The same problem occurs at the end of step 4.

Solution. We can orthogonalize the two branches by examining the rightmost cell of area i: its lower level is filled if we came from (2a), and empty if we came from (2b). This requires that we can tell where area i is, which means remembering the value i somehow. This is done by keeping the marks we placed on the scratch tape while finding area i, after steps (2a) and (2b), see Figure 5.

Cleanup confluence. The control flow confluence applies to cleanups in general: different order cleanups can lead to identical simulated tapes, as shown in Figure 6, and we can only tell these apart by remembering the order i of the cleanup. This is an inherent trait of the Hennie-Stearns simulation: the restoration of the standard reading in all the lower order blocks after a high order cleanup is essential for keeping the time complexity low.

Solution. The marked space on the scratch tape (a unary representation of 2^{i-1}) can be conserved, by always marking space towards the *right*, for simulating *both* left and right shifts. This ensures that we never move more than one cell to the left of the position of the scratch head at the start of a cleanup. After a cleanup has been performed, we then move the scratch head across the (conserved) markings and past the following blank square, to prepare for the next cleanup. Thus, the scratch tape is still used to perform linear time moves, but also stores a history of the previous markings, which defines the *orders* of the cleanups performed, see Figure 7. This is sufficient for reversibility.

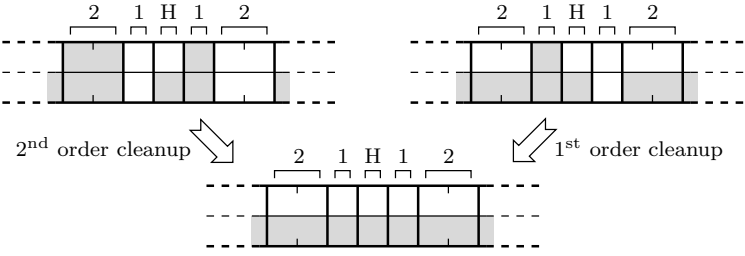


Fig. 6. Confluence of cleanups: 1st and 2nd order cleanups that lead to the same configuration, if the scratch tape (not shown) is fully cleared between cleanups. This shows the necessity of conserving some additional information between cleanups.

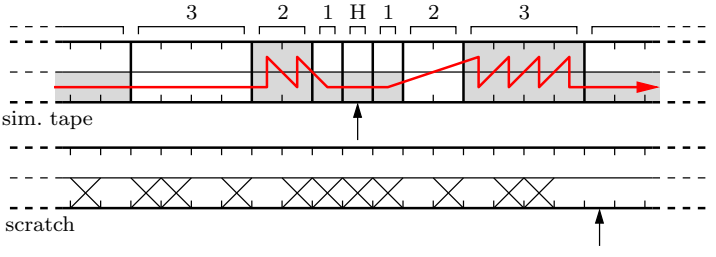


Fig. 7. The configuration of the 2-tape simulation after 4 left cleanups (of orders 1, 2, 1 and 3) and then 2 right cleanups (orders 1 and 2). When simulating multiple tapes, the orders of cleanups from the individual tapes will be interleaved on the scratch tape.

It is only the markings on the scratch tape that we cannot remove—the rest of the procedure works reversibly. While we do use a partial trace, importantly, it does *not* require an extra tape, like the Landauer embedding does. If desired (*e.g.* for function problems) this *order trace* can be removed by standard reversible computing methods, see Appendix A.

4.3 Reversible Time Complexity

None of the above solutions affect the time complexity of an ith order cleanup.

The novel delimiter-free layout requires that we implement the movement of symbols somewhat differently from the original version: in step (2a) we have to 'smooth out' left area i which is completely filled, and in step (4b) we have to 'fold up' right area i to fill it completely. By using a two-track scratch tape, this can be done with a constant number of passes, which conserves complexity. How this works is shown in Figure 8. Of course, this layout can also be used to obviate block delimiters in the irreversible simulation.

Conserving the marked space on the scratch tape during a cleanup adds no time overhead. Finally, passing over these marks to prepare the scratch tape for the next cleanup is also only linear in the number of marked cells.

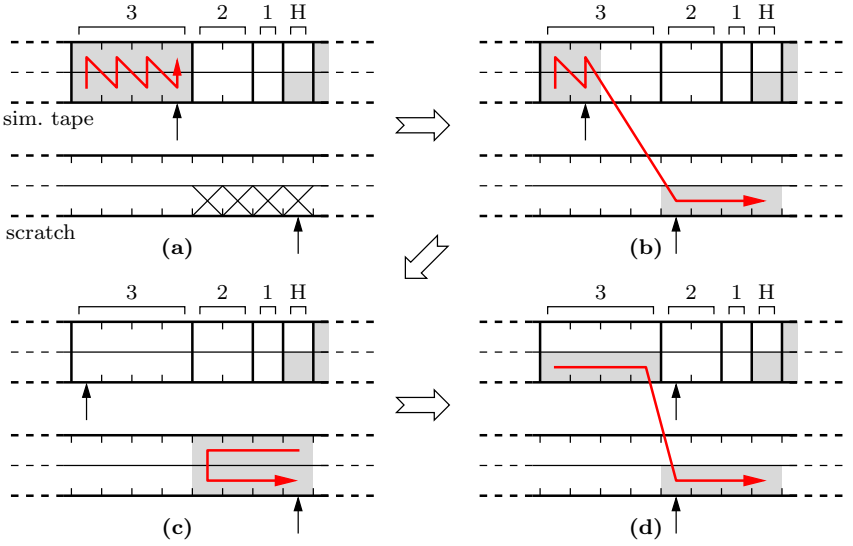


Fig. 8. Smoothing a tape reading in step (2a) of a 3rd order cleanup. To start with (a), the simulation tape has both levels of area 3 full, so we (b) move the first half into the lower level of the marked space on the scratch tape. We know when we have come halfway, because we run out of marks on the scratch tape. Then we (c) move the second half into the upper level on the scratch tape, and (d) move the upper level of the scratch level back into the lower level of area 3 of the simulation tape. This requires three passes of the area, regardless of its length.

We conclude that performing an ith order *reversible* cleanup takes time $\mathcal{O}(2^i)$. This matches the time complexity of the irreversible cleanup. Thus, the same argument used by Hennie and Stearns for the time complexity of their simulation applies to our reversible simulation of multitape RTMs.

Proposition 3. *A k-tape reversible Turing machine with running time $t(n)$ can be simulated by a 2-tape reversible Turing machine in $\mathcal{O}(t(n) \log t(n))$ time.*

This can, again, be used to elicit many further reversible complexity results. For instance, we now know that adding extra tapes to RTMs at most shaves off a logarithmic factor on any time bounds for multitape machines. Furthermore, the reversible Turing machines have an even tighter time hierarchy than before. Also, by Landauer embedding k-tape DTMs and applying the above 2-tape reversible simulation we have that $\bigcup_k \mathsf{DTIME}_k(t(n)) \subseteq \mathsf{RevTIME}_2(t(n) \log t(n))$.

5 Related Work

Most other work on reversible complexity has focused on space, time and energy trade-offs for reversible simulations of irreversible computations. Well-known results includes reversible simulation using pebble games, *e.g.* [14], and the (surprising) result that $\mathsf{RevSPACE}(s(n)) = \mathsf{DSPACE}(s(n))$ [9].

Kondacs and Watrous [7] proved that any DFA (*i.e.*, any regular language) can be simulated in linear time by a 2-way reversible DFA (essentially a read-only, one-tape RTM). As a consequence, $\mathsf{DTIME}(n) = \mathsf{REG} = \mathsf{RevTIME}(n)$, as noted in [13]. However, few other 'pure' reversible complexity results are known to the author.

As for multitape simulations, Pippenger and Fischer [12] showed that the 2-tape simulation of multitape DTMs can be made *oblivious*, such that the movement of the tape heads depends only on the length of the input.

6 Conclusion

In this paper we have shown that tape reduction of reversible multitape Turing machines (RTMs) incurs the same cost in time complexity as tape reduction for deterministic Turing machines in general. This can be leveraged to make statements about the (time) complexity of reversible machines more precise, and gives us almost direct access to a host of theorems about reversible complexity, especially for decision problems. We also provided a general method to eliminate stuck states from RTMs at no cost in complexity.

Adapting the Hennie-Stearns 2-tape simulation for reversibility required the use of a partial trace. No extra tape was needed for this; it can easily be removed from the final configuration, and it does not affect the time complexity of the simulation. However, it adds a time-dependent space overhead to the simulation that the 1-tape simulation does not. It is almost certainly possible to compress the trace at runtime, but avoiding a trace altogether appears doubtful.

We close with an open question. It is known that linear reversible and deterministic time are equal. Is reversible time equal to deterministic time *in general* for single-tape Turing machines, *i.e.*, is $\mathsf{RevTIME}(t(n)) = \mathsf{DTIME}(t(n))$?

Acknowledgements. The author would like to thank Michael Kirkedal Thomsen, Thomas Pécseli, and Robert Glück for comments on a draft of this paper.

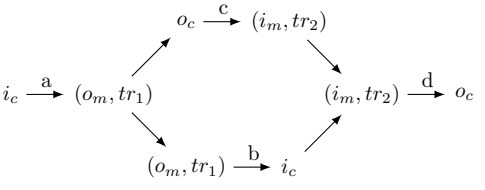
References

1. Axelsen, H.B., Glück, R.: A Simple and Efficient Universal Reversible Turing Machine. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 117–128. Springer, Heidelberg (2011)
2. Axelsen, H.B., Glück, R.: What Do Reversible Programs Compute? In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 42–56. Springer, Heidelberg (2011)
3. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. 17, 525–532 (1973)
4. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Trans. Amer. Math. Soc. 117, 285–306 (1965)
5. Hennie, F.: One-tape, off-line Turing machine computations. Inform. Control 8, 553–578 (1965)
6. Hennie, F., Stearns, R.E.: Two-tape simulation of multitape Turing machines. J. ACM 13(4), 533–546 (1966)

7. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: Proc. Foundations of Computer Science, pp. 66–75. IEEE (1997)
8. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. 5(3), 183–191 (1961)
9. Lange, K.J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. J. Comput. Syst. Sci. 60(2), 354–367 (2000)
10. Maass, W.: Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines. Trans. Amer. Math. Soc. 292(2), 675–693 (1985)
11. Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. Trans. IEICE E 72(3), 223–228 (1989)
12. Pippenger, N., Fischer, M.J.: Relations among complexity measures. J. ACM 26(2), 361–381 (1979)
13. Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time Turing machines. Theor. Comput. Sci. 411(1), 22–43 (2010)
14. Vitányi, P.: Time, space, and energy in reversible computing. In: Proc. Computing Frontiers, pp. 435–444. ACM (2005)
15. Yokoyama, T., Axelsen, H.B., Glück, R.: Optimizing reversible simulation of injective functions. J. Mult.-Val. Log. S. 18(1), 5–24 (2012)

A Removing the Trace

The order trace is an unwanted side effect of the simulation. There is the additional problem that we are not guaranteed that the simulated tapes have a 'smooth' contiguous structure at the end of the simulation, with the output string in only the lower levels of the areas. To solve this problem we can use a modified version of clean simulation of injective functions [15], as follows.



Run the simulation forwards (a) and extract a contiguous output copy o_c (placed on extra tracks) from the 'messy' output o_m. Run the simulation backwards (b), restoring the input in contiguous form, i_c, and removing the (partial) trace tr_1. Now run the simulation of the *inverse* machine on the contiguous output (c). The contiguous input i_c from (b) can be merged into the resulting 'messy' input i_m from (c). Finally, run the simulation of the inverse machine backwards (d), which removes the (partial) trace tr_2 resulting in only the contiguous output o_c.