Reversible Multi-head Finite Automata Characterize Reversible Logarithmic Space

Holger Bock Axelsen

DIKU, Dept. of Computer Science, University of Copenhagen, Denmark
www.diku.dk/~funkstar

LATA, A Coruña, 7 March 2012

Overview

- Motivation
- Informal acronyms: RMFAs and RTMs
- Simulating a logspace RTM with an RMFA
  - Encoding of configurations
  - Simulation of transitions
  - Read/write procedures
- Summing up

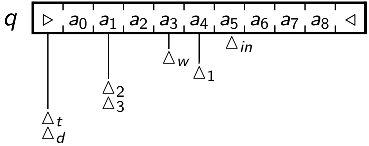
Motivation

Why this characterization?

- Recent open problem stated by Morita
  - Reversible PDAs are *weaker* than DPDAs
  - Reversible LBAs are *equivalent* to DLBAs
  - What about automata between these?
- $\mathcal{L}$(RMFA) is strong: palindromes, unary primes, DYCK(1)
- (D)MFAs characterize logspace very nicely
  - Logspace = constant amount of pointers
- Usual reversibilization trick won't work for RMFAs
  - No space for a history, or other *garbage data*
  - What to do if we *cannot* write at all?
- Nice use case for techniques from recent RTM work

Reversible Multi-head Finite Automata

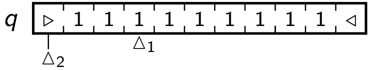
RMFA R

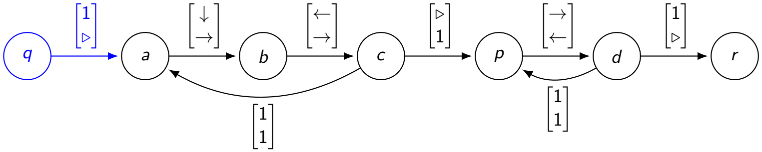


- Input word $w \in \Sigma^*$ in delimited form $\triangleright\, w \,\triangleleft$
- k heads (all start by pointing at $\triangleright$)
- Triple format transitions
  - symbol rules: $(q, [a, b, \triangleright, \cdots, c], p)$
  - move rules: $(q, [\leftarrow, \downarrow, \leftarrow, \cdots, \rightarrow], p)$
- Reversibility = forward + backward determinism
  - Fwd determinsm: $(q, s, \cdot)$ and $(q, t, \cdot)$ imply $s, t \in \Sigma^k$, $s \neq t$
  - Bwd determinsm: $(\cdot, s, p)$ and $(\cdot, t, p)$ imply $s, t \in \Sigma^k$, $s \neq t$

RMFA example

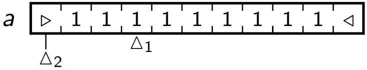
Doubling the unary counter in $\triangle_1$:

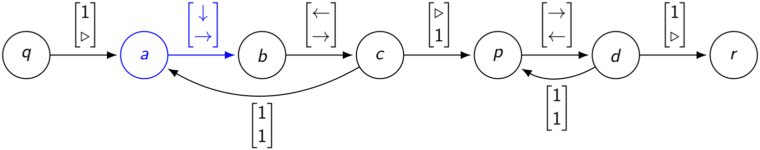




RMFA example

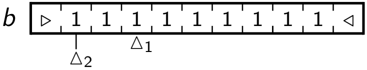
Doubling the unary counter in $\triangle_1$:

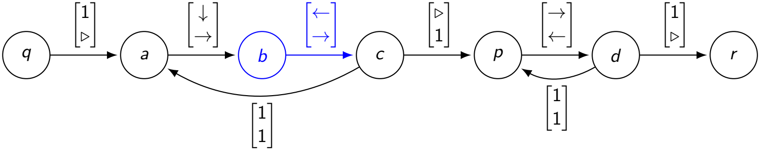




RMFA example

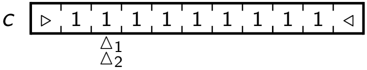
Doubling the unary counter in $\triangle_1$:

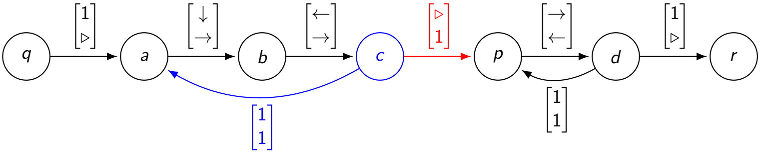




RMFA example

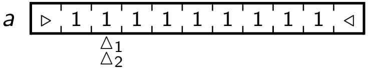
Doubling the unary counter in $\triangle_1$:

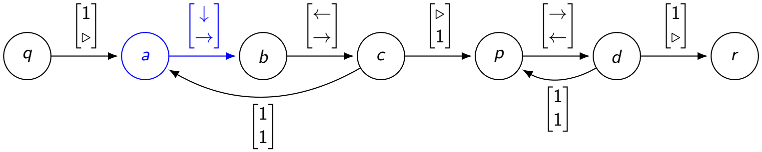




RMFA example

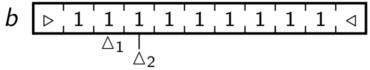
Doubling the unary counter in $\triangle_1$:

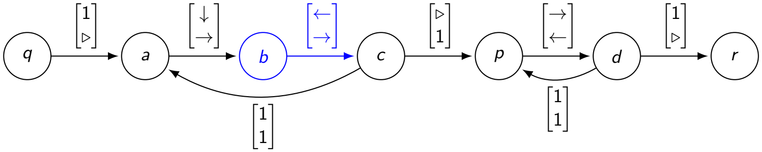




RMFA example

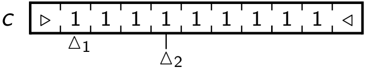
Doubling the unary counter in $\triangle_1$:

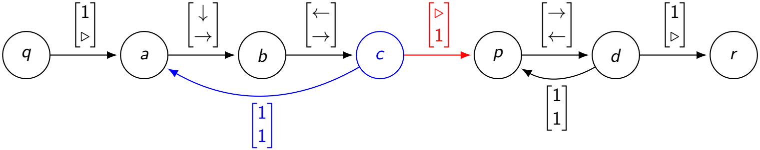




RMFA example

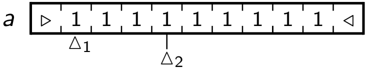
Doubling the unary counter in $\triangle_1$:

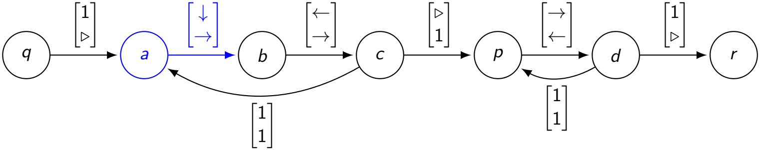




RMFA example

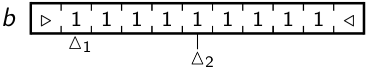
Doubling the unary counter in $\triangle_1$:

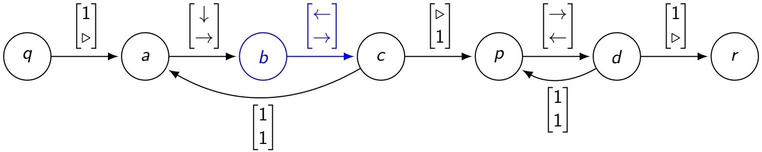




RMFA example

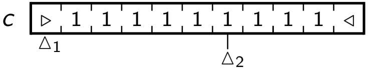
Doubling the unary counter in $\triangle_1$

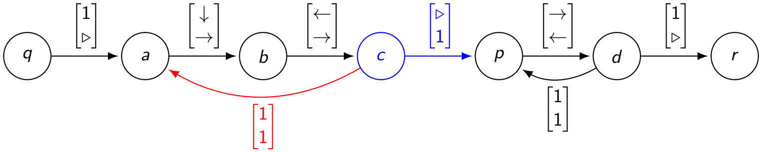




RMFA example

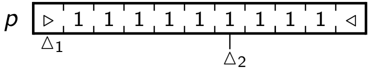
Doubling the unary counter in $\triangle_1$

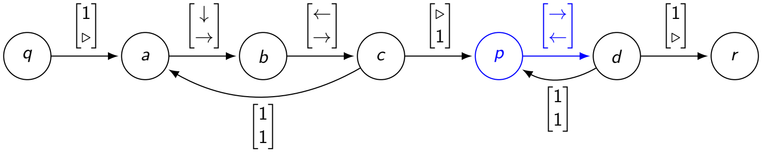




RMFA example

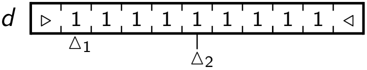
Doubling the unary counter in $\triangle_1$

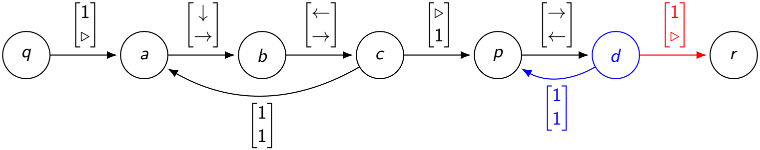




RMFA example

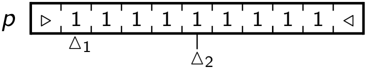
Doubling the unary counter in $\triangle_1$

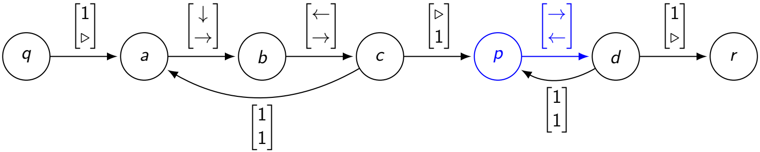




RMFA example

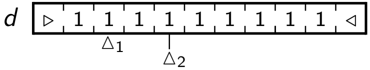
Doubling the unary counter in $\triangle_1$

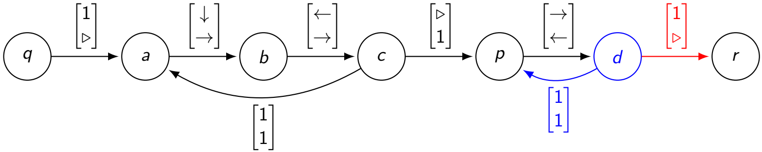




RMFA example

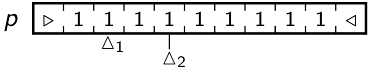
Doubling the unary counter in $\triangle_1$

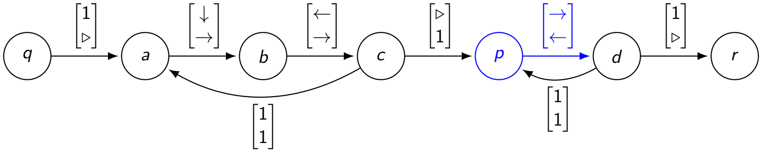




RMFA example

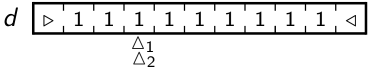
Doubling the unary counter in $\triangle_1$

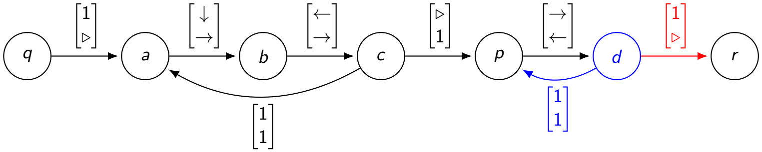




RMFA example

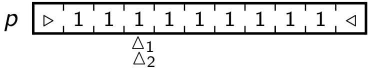
Doubling the unary counter in $\triangle_1$

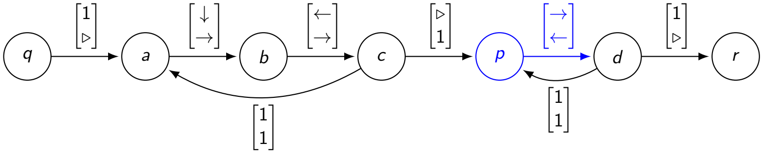




RMFA example

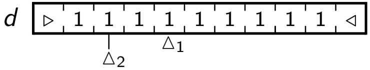
Doubling the unary counter in $\triangle_1$

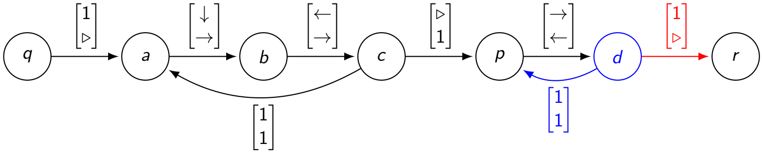




RMFA example

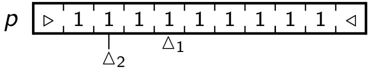
Doubling the unary counter in $\triangle_1$

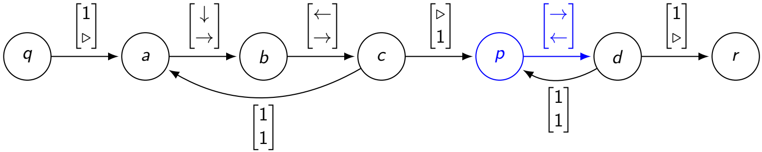




RMFA example

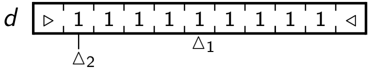
Doubling the unary counter in $\triangle_1$

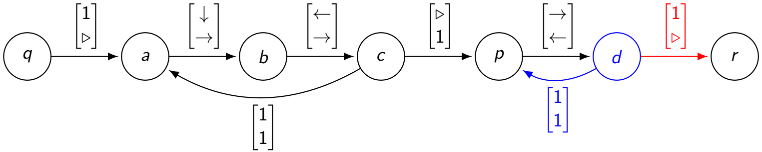




RMFA example

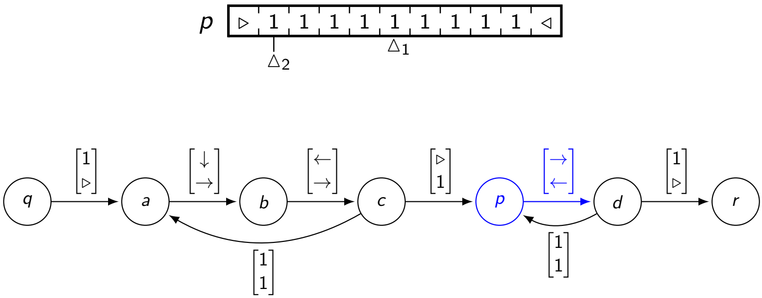
Doubling the unary counter in $\triangle_1$





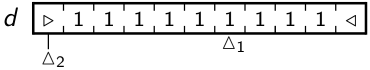
RMFA example

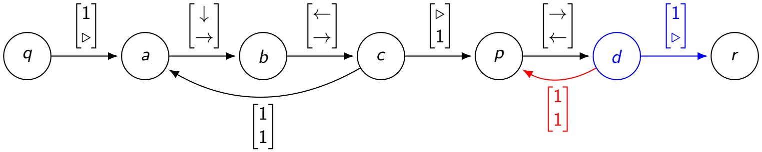
Doubling the unary counter in $\triangle_1$



RMFA example

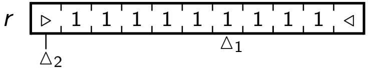
Doubling the unary counter in $\triangle_1$

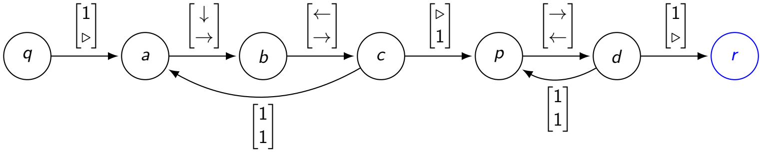




RMFA example

Doubling the unary counter in $\triangle_1$

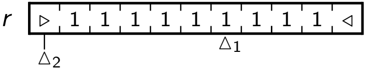




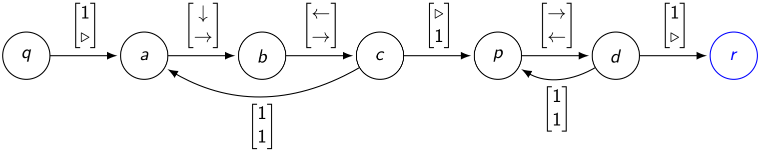
RMFA example: inversion

RMFAs are *invertible*: swap direction of transitions, swap $\leftarrow$ and $\rightarrow$

Doubling the unary counter in $\triangle_1$



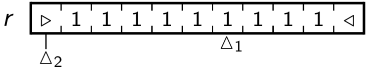
Invert:



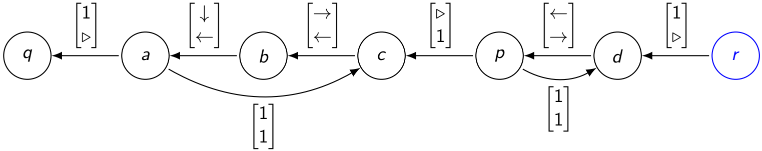
RMFA example: inversion

RMFAs are *invertible*: swap direction of transitions, swap $\leftarrow$ and $\rightarrow$

Doubling the unary counter in $\triangle_1$

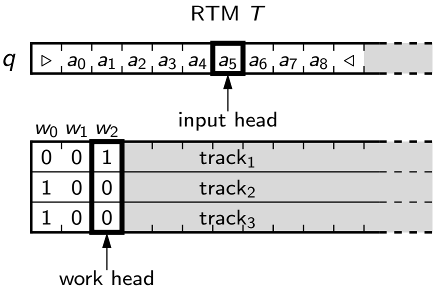


Invert:



This is an RMFA for halving the unary counter in $\triangle_1$

Reversible Turing Machines



- Input tape as before, but only one input head.
- Read/write work tape: one head, k binary tracks.
  - Space usage measured on the work tape.
- Triple format transitions, but symbol rules can now write on the work tape: $(q, [a, 100 \mapsto 011], p)$.
- Reversibility as before.

What do we know & what shall we prove

$$
\begin{array}{ccc}
\mathcal{L}(\text{RMFA}) & \overset{\text{Morita}}{=\!=} & \mathcal{L}(\text{DMFA}) \\
\| \text{ Here} & & \| \text{ Hartmanis'72} \\
\text{RevSPACE}(\log n) & \overset{\text{Lange } et\ al.\text{'00}}{=\!=} & \text{DSPACE}(\log n)
\end{array}
$$

Proving $\mathcal{L}(\text{RMFA}) = \text{RevSPACE}(\log n)$

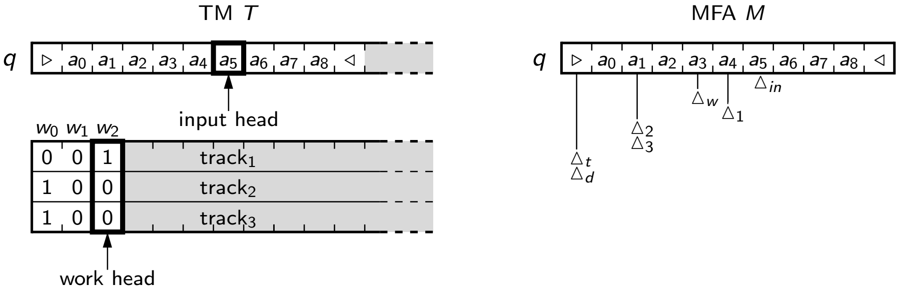
Will show $\text{RevSPACE}(\log n) \subseteq \mathcal{L}(\text{RMFA})$.

Proof: Simulate a logspace RTM with an RMFA.

- Encoding of RTM configurations
- Transition simulation
- Tape read/write simulation

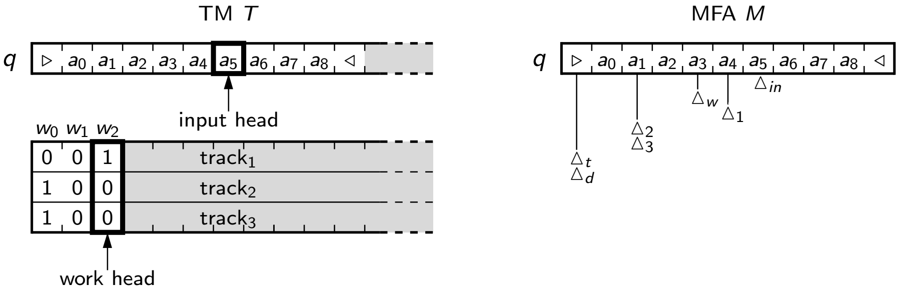
Encoding logspace TM configurations in MFAs

TMs use k binary tracks, strictly $\log n$ tape bounded.



- State q (in the TM) maps to q (in the MFA)
- $\triangle_{in}$ mirrors *input head* position
- $\triangle_w$ is $2^{\text{work head position}}$ places from $\triangleright$
- $\triangle_1, \triangle_2, \triangle_3, \ldots, \triangle_k$ simulates work tape *content* by track
- $\triangle_t, \triangle_d$ are ancillary heads for book-keeping

Encoding track content by head position



Track encoding: $\triangle_i$ points at a_n means track$_i$ contains $(\mathrm{bin}(n))^R$:

- track$_1$ contains '001', in reverse binary $100 = 4$,
- so $\triangle_1$ points at a_4, encoding 4 in unary.

To read work cell w_2 we must consult all of $\triangle_1, \triangle_2, \triangle_3$

Simulating RTM transitions

By determinism, the transitions going out of source state q are

- all symbol rules, or
- a single move rule.

We'll tackle these in turn.

Simulating symbol transitions $(q, [\cdot, \cdot \mapsto \cdot], \cdot)$

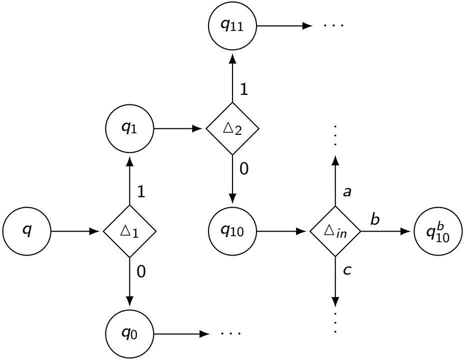
To simulate a symbol transition $(q, [\cdot, \cdot \mapsto \cdot], \cdot)$ we need

- to read the input tape symbol (easy)
- to read the work tape symbol (not so easy)
- to write a new work tape symbol (easy)
- to go into target state (not so easy)

Work tape symbol is encoded across all track heads by their *position alone*, not the local symbol.

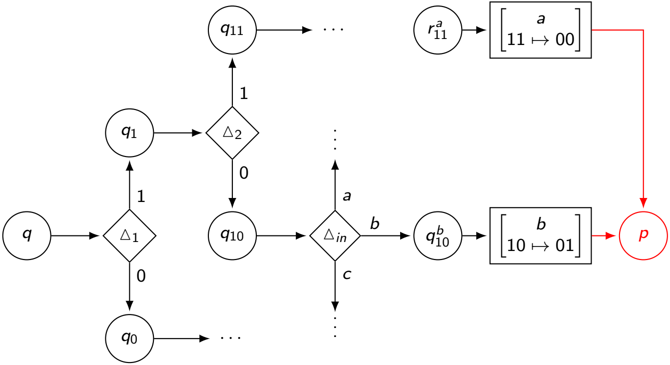
Use a *staged reading* of the symbols, store in the state.

Simulating RTM symbol rules $(q, [\cdot, \mapsto \cdot], \cdot)$



(Assuming we have a read procedure for track heads $\triangle_i$.)

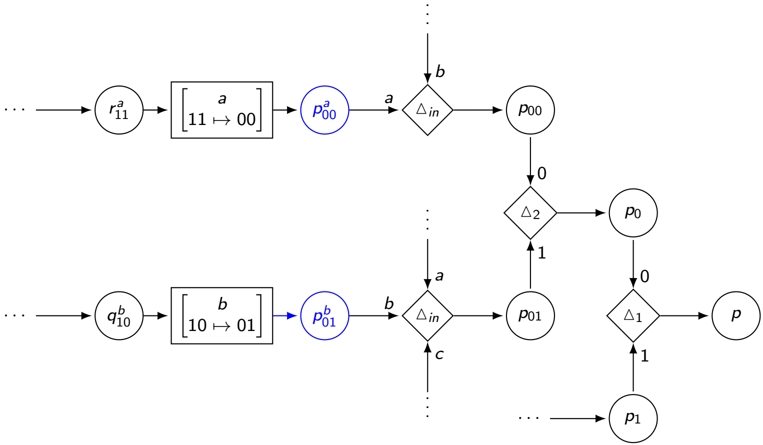
Simulating RTM symbol rules $(q, [b, 10 \mapsto 01], p)$



Doesn't work: there can be *other* symbol transitions targeting p, say, $(r, [a, 11 \mapsto 00], p)$. Direct links to p break reversibility.

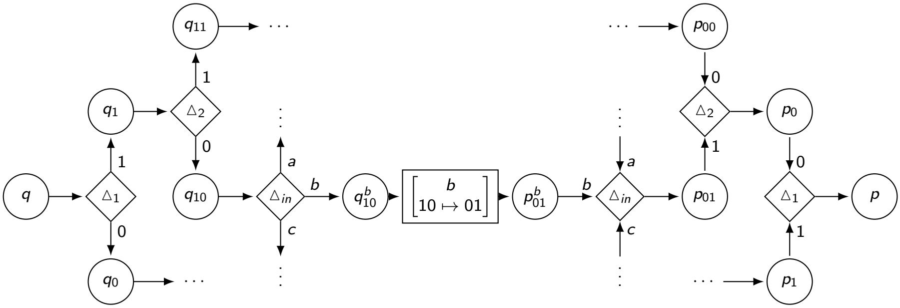
Solution: Exploit reversibility of the RTM. *No other transition targeting p can write both b and 01 to the tapes.*

Simulating RTM symbol rules $(\cdot, [\cdot, \cdot \mapsto \cdot], p)$



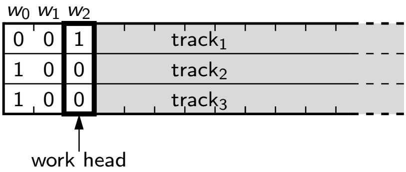
Intuition: Link up the appropriate leaves of decision trees of *source* states (r, q) with leaves of inverse decision tree for *target* state p.

Simulating RTM symbol rules $(q, [b, 10 \mapsto 01], p)$



Subpart of symbol rule simulation for $(q, [b, 10 \mapsto 01], p)$.

Reading procedure



1. Each track head encodes just 1 bit from each work tape cell.
2. If the RTM work head points to cell w_p, then the RMFA work head $\triangle_w$ is 2^p cells to the right of $\triangleright$.

Idea: The pth bit of n in binary is

$$(n \text{ div } 2^p) \bmod 2\,.$$

Reading procedure 2

Idea: The pth bit of n in binary is

$$(n \text{ div } 2^p) \bmod 2\,.$$

most sign.	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
least sign.	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Reading procedure: Read off pth bit from track i, by repeatedly 'subtracting' 2^p ($\triangle_w$'s offset) from $\triangle_i$. $\triangle_i$'s original position is conserved in $\triangle_t$, and the original position of $\triangle_w$ in $\triangle_d$.

Reading procedure 2

Idea: The pth bit of n in binary is

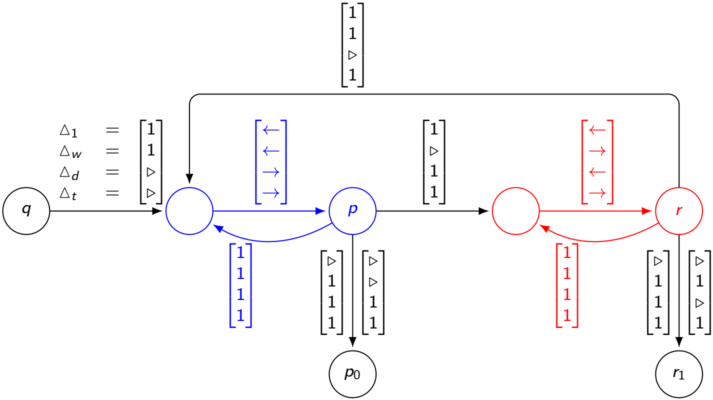
$$(n \text{ div } 2^p) \bmod 2\,.$$

most sign.	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
least sign.	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\triangleright$											$\triangle_2$					

Reading procedure: Read off pth bit from track i, by repeatedly 'subtracting' 2^p ($\triangle_w$'s offset) from $\triangle_i$. $\triangle_i$'s original position is conserved in $\triangle_t$, and the original position of $\triangle_w$ in $\triangle_d$.

Example: $\triangle_2$ point at a_{10}, which encodes 1010. $\triangle_w$ points at w_2, so repeatedly subtract $2^2 = 4$ from 10, until we hit $\triangleright$.

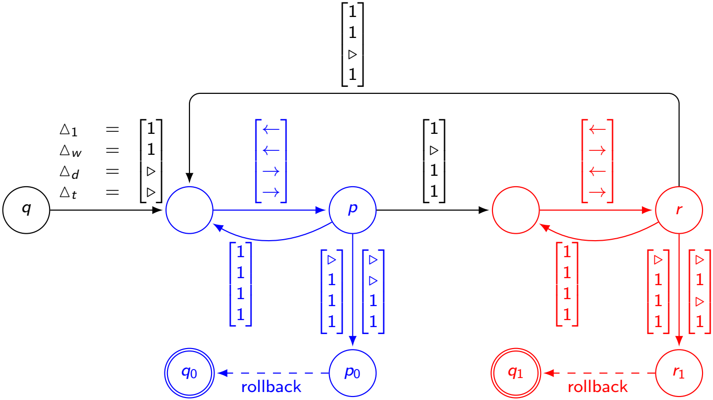
Reversible reading procedure for $\triangle_1$



Blue loop: subtract $\triangle_w$ from $\triangle_1$, believing the bit is 0.

Red loop: subtract $\triangle_w$ from $\triangle_1$, believing the bit is 1.

Reversible reading procedure for $\triangle_1$



Problem: head positions are not conserved.

Solution: *rollbacks* are *inverse* copies of the reading procedure, restoring all heads to their previous positions, *reversibly!*

Reversible writing procedure

An RTM symbol rule such as $(q, (a, 1010 \mapsto 0011), p)$ define *explicit* bit flips. Flipping the pth bit encoded on track i is simple:

- $0 \mapsto 1$: move $\triangle_i$ 2^p positions to the *right*
- $1 \mapsto 0$: move $\triangle_i$ 2^p positions to the *left*

Example: Let $\triangle_2$ point at a_{10}, which encodes 1010. To flip the 3rd bit from 0 to 1 we move $\triangle_2$ 2^2 right. It then points at a_{14}, encoding 1110.

Although overwriting symbols *in general* is irreversible, overwriting a *specific* known symbol is not.

Simulate move rules

Simulating an RTM move $(q, [\leftarrow, \rightarrow], p)$ is considerably easier.

The RMFA does as follows.

- From q,
- $\triangle_{in}$ mirrors input head,
- if work tape head moves:
  - right, then *double* $\triangle_w$
  - left, then *halve* $\triangle_w$ (with inverse doubling procedure)
- go into state p.

Note: We do *not* have to worry about about other transitions targeting p, because by reversibilty there *are no others!*

Summary

We now have:

- Encoding from RTM to RMFA configuration
- Simulation of RTM transitions in the RMFA

So we're done with showing $\mathrm{RevSPACE}(\log n) \subseteq \mathcal{L}(\mathrm{RMFA})$.

The reverse inclusion, $\mathcal{L}(\mathrm{RMFA}) \subseteq \mathrm{RevSPACE}(\log n)$ is considerably easier, so

$$\mathrm{RevSPACE}(\log n) = \mathcal{L}(\mathrm{RMFA})\,.$$

Future work: Other reversible automata models, 2-way vs. 1-way.

www.reversible-computation.org


RC2012

4th Workshop on
Reversible Computation (RC)
July 2nd -3rd 2012, Copenhagen, Denmark

- Home
- Call for Papers
- Program Committee
- Paper Submission
- Location
- Previous Editions
- Contact

Home

Welcome to the
4th Workshop on Reversible Computation

July 2nd-3rd, 2012, Copenhagen, Denmark

Deadline extended:
- Abstract Submission: March 9th, 2012
- Submission Deadline: March 16th, 2012

The Workshop on Reversible Computation will bring together researchers from computer science, mathematics, engineering, and physics to discuss new developments and directions for future research in the emerging area of Reversible Computation. This particularly includes applications of reversibility in quantum computation. Research papers, tutorials, tool demonstrations, and work-in-progress reports are within the scope of the workshop.

The 4th Workshop on Reversible Computation will take place on July 2nd and 3rd, 2012 in Copenhagen, Denmark and is organized by the University of Copenhagen. Accepted papers will appear in the preliminary proceedings at the workshop. Furthermore, authors of selected papers will be invited after the workshop to prepare a final version of their paper to be published in Springer's Lecture Notes in Computer Science (LNCS).

The Workshop will be held in parallel with the European Conference on Modelling Foundations and Applications.

Download
Call for Paper 2012

Important Dates:

Abstract Submission:
Fir, March 9th, 2012

Submission Deadline:
Fri, March 16th, 2012

Notification to Authors:
Mon, May 7th, 2012

Final Version:
Fri, June 1st, 2012

Workshop:
Mon-Tue July 2nd-3rd, 2012

Workshop Organizer

Holger Bock Axelsen
University of Copenhagen

Department of Computer Science
Universitetsparken 1
2100 Denmark
info@reversible-computation.org

Impress - Contact - Webmaster

University of Copenhagen | info@reversible-computation.org | e

4th edition on July 2nd-3rd in Copenhagen, Denmark