Mechanized Undecidability of Halting Problems for Reversible Machines

Final Talk

Hizbullah A. A. Jabbar
Advisor: Dr. Andrej Dudenhefner
Supervisor: Prof. Gert Smolka

Saarland University

29/08/2022

Outline

Outline

Reversible Machines

Reversible Machines

- A reversible machines are those whose computations can be retraced back in time (Kari and Ollinger, 2008).

Reversible Machines

- A reversible machines are those whose computations can be retraced back in time (Kari and Ollinger, 2008).
- Injective step relations.

Reversible Machines

- A reversible machines are those whose computations can be retraced back in time (Kari and Ollinger, 2008).
- Injective step relations.
- Interests in reversible machines stem from Landauer's Principle (Landauer, 1961).

Reversible Machines

- A reversible machines are those whose computations can be retraced back in time (Kari and Ollinger, 2008).
- Injective step relations.
- Interests in reversible machines stem from Landauer's Principle (Landauer, 1961).
- See (Bennett, 2003) for a more thorough treatment on Landauer's Principle.

Goal

The goal of this thesis is to mechanize in Coq the (un)-decidability of halting problems for:

- Reversible FRACTRANs
- Reversible 2-counter machines
- Weakly-reversible 2-dimensional cellular automata

(Extensional) Reversibility

(Extensional) Reversibility

A machine M is *extensionally reversible* iff for all its configurations s, t, and u, if M steps from s to u and from t to u then $s = t$.

(Extensional) Reversibility

A machine M is *extensionally reversible* iff for all its configurations s, t, and u, if M steps from s to u and from t to u then $s = t$.

Exactly the dual notion of determinism.

(Extensional) Reversibility

A machine M is *extensionally reversible* iff for all its configurations s, t, and u, if M steps from s to u and from t to u then $s = t$.

Exactly the dual notion of determinism.

A machine M is *weakly reversible* iff for all its configurations s, t, and u, if M steps from s to u and from t to u and u is not halting then $s = t$.

Outline

FRACTRAN

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 1

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 1

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 16

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[{\color{red}\frac{2}{3}}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, {\color{red}\frac{1}{4}}, \frac{5}{6}]$ configuration: 4

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 1

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 16

$[\frac{2}{3}, {\color{red}\frac{1}{4}}, \frac{5}{6}]$ configuration: 16

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 1

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 16

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 16

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

FRACTRAN

FRACTRAN is a simple computation model with undecidable halting problems (Conway, 1987).

A FRACTRAN program is a list of fractions whose configurations are natural numbers.

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 6

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 1

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 16

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 16

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 4

$[\frac{2}{3}, \frac{1}{4}, \frac{5}{6}]$ configuration: 1

FRACTRAN

Theorem

Reversible FRACTRAN programs have decidable halting problems.

Proof.

The key observation here is that reversible FRACTRAN programs essentially contain at most one fraction[1], for which one can construct halting deciders for. □

[1] with insights from a private communication with Dominique Larchey-Wendling

Outline

Counter Machines

Counter Machines

- Counter machines are one of the most well-studied computation models with undecidable halting problems (Minsky, 1967).

Counter Machines

- Counter machines are one of the most well-studied computation models with undecidable halting problems (Minsky, 1967).
- A counter machine is a list of instructions (from some instruction set) whose configurations are pairs $(i, \overline{v})$ of addresses i and counters values $\overline{v}$.

Counter Machines

Counter Machines

- Two existing counter machine formalizations in the Coq Library of Undecidability Proofs (Forster et al., 2020): MM and MMA/CM.

Counter Machines

- Two existing counter machine formalizations in the Coq Library of Undecidability Proofs (Forster et al., 2020): MM and MMA/CM.
- They are essentially lists of increment and decrement instructions.

Counter Machines

- Two existing counter machine formalizations in the Coq Library of Undecidability Proofs (Forster et al., 2020): MM and MMA/CM.
- They are essentially lists of increment and decrement instructions.
- INC x at address i increments the counter x and jumps to $i + 1$.

Counter Machines

- Two existing counter machine formalizations in the Coq Library of Undecidability Proofs (Forster et al., 2020): MM and MMA/CM.
- They are essentially lists of increment and decrement instructions.
- INC x at address i increments the counter x and jumps to $i + 1$.
- For MM, DEC x j at address i jumps to j if the counter x contains zero (leaving it unchanged), otherwise decrements the counter x and jumps to $i + 1$.

Counter Machines

- Two existing counter machine formalizations in the Coq Library of Undecidability Proofs (Forster et al., 2020): MM and MMA/CM.
- They are essentially lists of increment and decrement instructions.
- INC x at address i increments the counter x and jumps to $i + 1$.
- For MM, DEC $x\ j$ at address i jumps to j if the counter x contains zero (leaving it unchanged), otherwise decrements the counter x and jumps to $i + 1$.
- For MMA/CM, DEC $x\ j$ at address i decrements the counter x and jumps to j if it contains a positive number, otherwise it leaves x unchanged and jumps to $i + 1$.

Counter Machines

Counter Machines

- Two-counter MMs have decidable halting problems (Dudenhefner, 2022).

Counter Machines

- Two-counter MMs have decidable halting problems (Dudenhefner, 2022).
- Reversible two-counter MMA/CMs also have decidable halting problems (Dudenhefner, 2022).

Counter Machines

- Two-counter MMs have decidable halting problems (Dudenhefner, 2022).
- Reversible two-counter MMA/CMs also have decidable halting problems (Dudenhefner, 2022).
- Hence we use Morita's counter machines (Morita, 1996).

Morita's Counter Machine

Morita's Counter Machine

- A Morita's counter machine (Morita, 1996) have 5 operations: increment, decrement, unconditional jump, zero test, and positive test.

Morita's Counter Machine

- A Morita's counter machine (Morita, 1996) have 5 operations: increment, decrement, unconditional jump, zero test, and positive test.

Morita's Counter Machine

- A Morita's counter machine (Morita, 1996) have 5 operations: increment, decrement, unconditional jump, zero test, and positive test.
- MM and MMA/CMs are deterministic by constructions: only one instruction per address.

Morita's Counter Machine

- A Morita's counter machine (Morita, 1996) have 5 operations: increment, decrement, unconditional jump, zero test, and positive test.
- MM and MMA/CMs are deterministic by constructions: only one instruction per address.
- But Morita's counter machines can be non-deterministic.

Morita's Counter Machine

Morita's Counter Machine

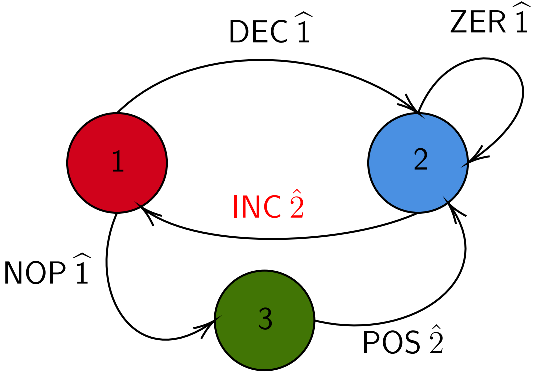
A Morita's counter machine is a list of instructions (p, x, i, j)

Morita's Counter Machine

A Morita's counter machine is a list of instructions (p, x, i, j)
e.g. $(\mathsf{INC}, \hat{2}, \color{blue}{2}, \color{red}{1})$.

Morita's Counter Machine

A Morita's counter machine is a list of instructions (p, x, i, j) e.g. $(\mathsf{INC}, \hat{2}, {\color{blue}2}, {\color{red}1})$.



Morita's construction

Morita's construction

- Morita's construction (Morita, 1996) is a reduction from the halting problems of deterministic k-Morita's counter machines to the halting problems of deterministic and reversible 2-Morita's counter machines.

Morita's construction

- Morita's construction (Morita, 1996) is a reduction from the halting problems of deterministic k-Morita's counter machines to the halting problems of deterministic and reversible 2-Morita's counter machines.
- We partially mechanize Morita's construction for 2-Morita's counter machines.

Morita's construction

- Morita's construction (Morita, 1996) is a reduction from the halting problems of deterministic k-Morita's counter machines to the halting problems of deterministic and reversible 2-Morita's counter machines.
- We partially mechanize Morita's construction for 2-Morita's counter machines.

Morita's construction

- Morita's construction (Morita, 1996) is a reduction from the halting problems of deterministic k-Morita's counter machines to the halting problems of deterministic and reversible 2-Morita's counter machines.
- We partially mechanize Morita's construction for 2-Morita's counter machines.

Need to characterize reversible Morita's counter machines syntatically.

Intensional Reversibility

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- $(\text{ZER}, \hat{1}, 1, 1)$ and $(\text{POS}, \hat{2}, 1, 1)$ overlap in range, but $(\text{ZER}, \hat{1}, 1, 1)$ and $(\text{POS}, \hat{1}, 1, 1)$ do not.

Intensional Reversibility

- Morita proposed a syntactic criterion for reversibility using the so-called *range overlap*.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- $(\mathsf{ZER}, \hat{1}, 1, 1)$ and $(\mathsf{POS}, \hat{2}, 1, 1)$ overlap in range, but $(\mathsf{ZER}, \hat{1}, 1, 1)$ and $(\mathsf{POS}, \hat{1}, 1, 1)$ do not.
- A Morita's counter machine is *intensionally reversible* if none of its instructions overlap in range.

Intensional Reversibility

Intensional Reversibility

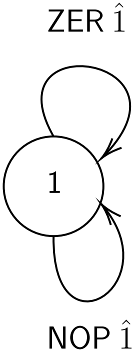
Lemma

Intensional reversibility implies extensional reversibility.

Intensional Reversibility

Lemma

Intensional reversibility implies extensional reversibility.



Morita's construction in steps

Morita's construction in steps

1. Reduce the indegree of each address to two or less.

Morita's construction in steps

1. Reduce the indegree of each address to two or less.
2. For each pair of instructions that overlap in range, record which instructions were executed as a binary number using two extra counters.

Morita's construction in steps

1. Reduce the indegree of each address to two or less.
2. For each pair of instructions that overlap in range, record which instructions were executed as a binary number using two extra counters.
3. Compression: reduce the number of counters back to two using Gödel numbering.

Morita's construction in steps

1. Reduce the indegree of each address to two or less.
2. For each pair of instructions that overlap in range, record which instructions were executed as a binary number using two extra counters.
3. Compression: reduce the number of counters back to two using Gödel numbering.

We partially mechanize Morita's construction by using graph representations.

Graph representation

Graph representation

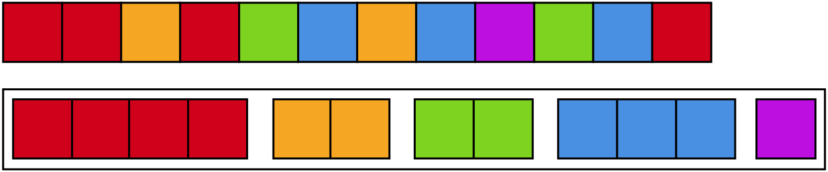
Recall that a Morita's counter machine is formalized as a list of instructions (p, x, i, j).

Graph representation

Recall that a Morita's counter machine is formalized as a list of instructions (p, x, i, j).

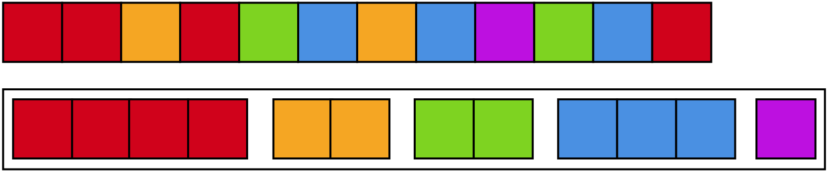
Graph representation

Recall that a Morita's counter machine is formalized as a list of instructions (p, x, i, j).



Graph representation

Recall that a Morita's counter machine is formalized as a list of instructions (p, x, i, j).

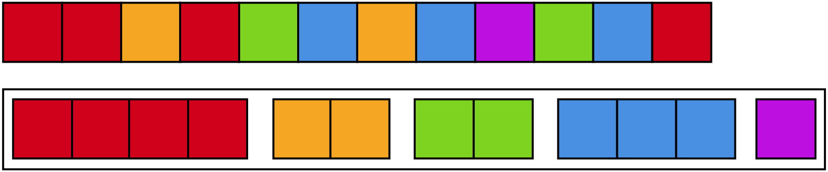


Lemma

If every sublists is intensionally reversible, then the whole graph is intensionally reversible.

Graph representation

Recall that a Morita's counter machine is formalized as a list of instructions (p, x, i, j).



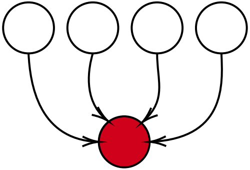
Lemma

If every sublists is intensionally reversible, then the whole graph is intensionally reversible.

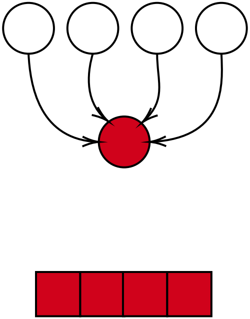
Each step of Morita's construction can then be implemented as a simple map or flat-map.

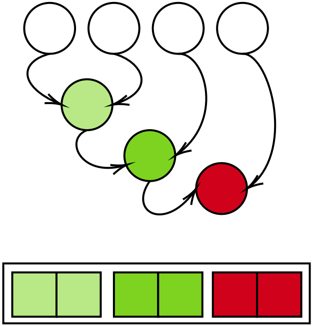
Reducing indegree

Reducing indegree

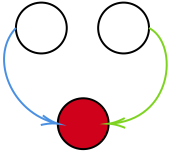


Reducing indegree

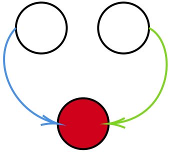




Recording history

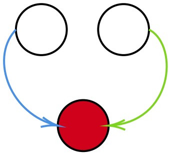


Recording history



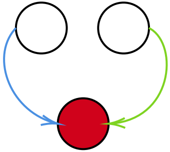
- Two extra counters were added: $\hat{1}$ to store history and $\hat{2}$ as an auxillary counter.

Recording history



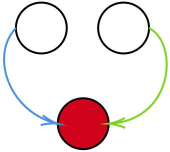
- Two extra counters were added: $\hat{1}$ to store history and $\hat{2}$ as an auxillary counter.
- Suppose that the current history value is n.

Recording history



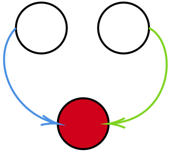
- Two extra counters were added: $\hat{1}$ to store history and $\hat{2}$ as an auxillary counter.
- Suppose that the current history value is n.
- If the left instruction was executed, store $2n$ at counter $\hat{1}$.

Recording history



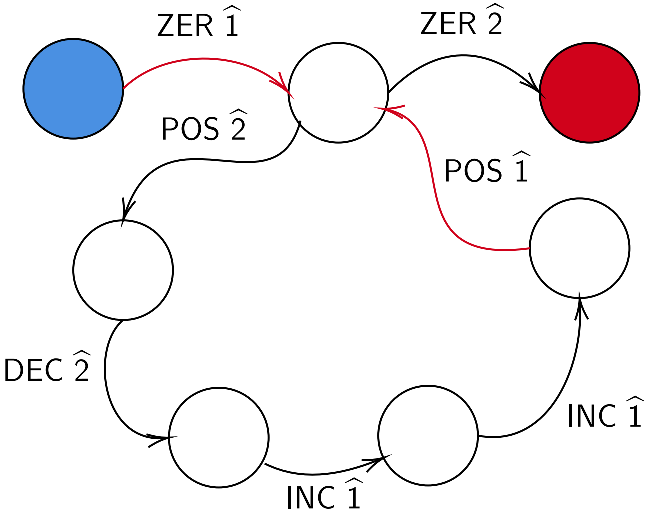
- Two extra counters were added: $\hat{1}$ to store history and $\hat{2}$ as an auxillary counter.
- Suppose that the current history value is n.
- If the left instruction was executed, store $2n$ at counter $\hat{1}$.
- Otherwise, the right instruction was executed, store $2n+1$ at counter $\hat{1}$.

Recording history



- Two extra counters were added: $\hat{1}$ to store history and $\hat{2}$ as an auxillary counter.
- Suppose that the current history value is n.
- If the left instruction was executed, store $2n$ at counter $\hat{1}$.
- Otherwise, the right instruction was executed, store $2n+1$ at counter $\hat{1}$.
- Need a way to construct reversible loops.

Recording history



Compression

Compression

- Reducing the number of counters back to two via Gödel encoding is a well-understood process.

Compression

- Reducing the number of counters back to two via Gödel encoding is a well-understood process.
- Instead of working with counters $[v_1, v_2, v_3, \ldots, v_k]$, one works with $[2^{v_1} 3^{v_2} 5^{v_5} \ldots p_k^{v_k}, 0]$.

Compression

- Reducing the number of counters back to two via Gödel encoding is a well-understood process.
- Instead of working with counters $[v_1, v_2, v_3, \ldots, v_k]$, one works with $[2^{v_1} 3^{v_2} 5^{v_5} \ldots p_k^{v_k}, 0]$.
- Thus incrementing the first counter becomes multiplication by 2, for example.

Compression

- Reducing the number of counters back to two via Gödel encoding is a well-understood process.
- Instead of working with counters $[v_1, v_2, v_3, \ldots, v_k]$, one works with $[2^{v_1}3^{v_2}5^{v_5}\ldots p_k^{v_k}, 0]$.
- Thus incrementing the first counter becomes multiplication by 2, for example.
- The crucial point here is to preserve reversibility,

Compression

- Reducing the number of counters back to two via Gödel encoding is a well-understood process.
- Instead of working with counters $[v_1, v_2, v_3, \ldots, v_k]$, one works with $[2^{v_1} 3^{v_2} 5^{v_5} \ldots p_k^{v_k}, 0]$.
- Thus incrementing the first counter becomes multiplication by 2, for example.
- The crucial point here is to preserve reversibility, but we already know how to construct loops that preserve reversibility.

Compression

- Reducing the number of counters back to two via Gödel encoding is a well-understood process.
- Instead of working with counters $[v_1, v_2, v_3, \ldots, v_k]$, one works with $[2^{v_1} 3^{v_2} 5^{v_5} \ldots p_k^{v_k}, 0]$.
- Thus incrementing the first counter becomes multiplication by 2, for example.
- The crucial point here is to preserve reversibility, but we already know how to construct loops that preserve reversibility.
- We did not mechanize this step due to time constraint.

Outline

Cellular Automata

Cellular Automata

- Cellular automata represent massively parallel computations.

Cellular Automata

- Cellular automata represent massively parallel computations.
- Famous example: Wolfram's Rule 110, which has been shown to be computationally universal (Wolfram, 2002).

Cellular Automata

- Cellular automata represent massively parallel computations.
- Famous example: Wolfram's Rule 110, which has been shown to be computationally universal (Wolfram, 2002).
- A cellular automaton is a characterized by its local update rule defined over a neighborhood

Cellular Automata

- Cellular automata represent massively parallel computations.
- Famous example: Wolfram's Rule 110, which has been shown to be computationally universal (Wolfram, 2002).
- A cellular automaton is a characterized by its local update rule defined over a neighborhood whose simultaneous applications determine its next configuration.

Cellular Automata

- Cellular automata represent massively parallel computations.
- Famous example: Wolfram's Rule 110, which has been shown to be computationally universal (Wolfram, 2002).
- A cellular automaton is a characterized by its local update rule defined over a neighborhood whose simultaneous applications determine its next configuration.
- The local update rule is applied homogenously, globally.

One-dimensional Cellular Automata

One-dimensional Cellular Automata

- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where

One-dimensional Cellular Automata

- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet,

One-dimensional Cellular Automata

- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet, $f : \Sigma^{2r+1} \to \Sigma$ is a local update function, and

One-dimensional Cellular Automata

- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet, $f : \Sigma^{2r+1} \to \Sigma$ is a local update function, and r is the neighborhood radius.

One-dimensional Cellular Automata

- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet, $f : \Sigma^{2r+1} \to \Sigma$ is a local update function, and r is the neighborhood radius.
- Its configurations are functions $s : \mathbb{Z} \to \Sigma$, which can be thought of as arrays of cells.

One-dimensional Cellular Automata

- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet, $f : \Sigma^{2r+1} \to \Sigma$ is a local update function, and r is the neighborhood radius.
- Its configurations are functions $s : \mathbb{Z} \to \Sigma$, which can be thought of as arrays of cells.
- There is a quiescent letter: $q \in \Sigma$ such that $f(q, q, \ldots, q) = q$.

One-dimensional Cellular Automata

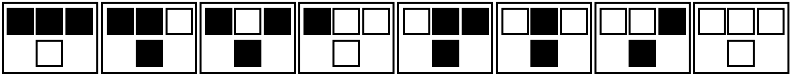
- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet, $f : \Sigma^{2r+1} \to \Sigma$ is a local update function, and r is the neighborhood radius.
- Its configurations are functions $s : \mathbb{Z} \to \Sigma$, which can be thought of as arrays of cells.
- There is a quiescent letter: $q \in \Sigma$ such that $f(q, q, \ldots, q) = q$.
- A CA1 configuration is spatially-finite iff beyond some bound $\pm n$, every cell contains a quiescent letter.

One-dimensional Cellular Automata

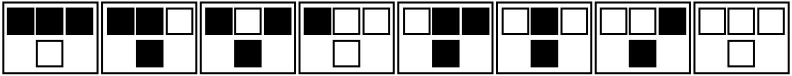
- A one-dimensional cellular automaton (CA1) is a triple (Σ, f, r) where Σ is a finite alphabet, $f : \Sigma^{2r+1} \to \Sigma$ is a local update function, and r is the neighborhood radius.
- Its configurations are functions $s : \mathbb{Z} \to \Sigma$, which can be thought of as arrays of cells.
- There is a quiescent letter: $q \in \Sigma$ such that $f(q, q, \ldots, q) = q$.
- A CA1 configuration is spatially-finite iff beyond some bound $\pm n$, every cell contains a quiescent letter.
- A CA1 configuration is halting if it is filled with quiescent letters.

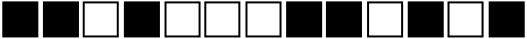
Rule 110

Rule 110

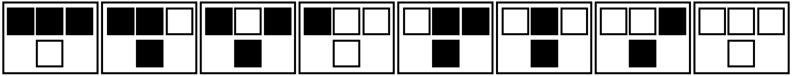


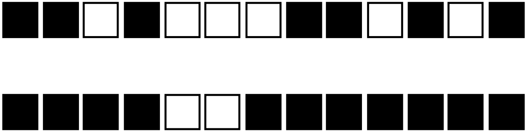
Rule 110





Rule 110





CA1, Mechanized

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.
- Instead, cells contain $\mathcal{O}(\Sigma)$ instead of Σ:

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.
- Instead, cells contain $\mathcal{O}(\Sigma)$ instead of Σ: a configuration is halting if a cell contains $\emptyset$.

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.
- Instead, cells contain $\mathcal{O}(\Sigma)$ instead of Σ: a configuration is halting if a cell contains $\emptyset$.
- Consequently, local update rules return $\mathcal{O}(\Sigma)$ instead of Σ.

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.
- Instead, cells contain $\mathcal{O}(\Sigma)$ instead of Σ: a configuration is halting if a cell contains $\emptyset$.
- Consequently, local update rules return $\mathcal{O}(\Sigma)$ instead of Σ.
- We consider only spatially-finite configurations.

CA1, Mechanized

- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.
- Instead, cells contain $\mathcal{O}(\Sigma)$ instead of Σ: a configuration is halting if a cell contains $\emptyset$.
- Consequently, local update rules return $\mathcal{O}(\Sigma)$ instead of Σ.
- We consider only spatially-finite configurations.
- Reduction from binary Turing Machines is relatively straightforward:

CA1, Mechanized

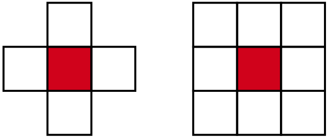
- We consider CA1s with neighborhood radius 1.
- Defining termination using quiescent configurations trivially breaks reversibility due to self-loops.
- Instead, cells contain $\mathcal{O}(\Sigma)$ instead of Σ: a configuration is halting if a cell contains $\emptyset$.
- Consequently, local update rules return $\mathcal{O}(\Sigma)$ instead of Σ.
- We consider only spatially-finite configurations.
- Reduction from binary Turing Machines is relatively straightforward: one needs to also track where the head of the Turing Machine is.

Two-dimensional Cellular Automata

Two-dimensional Cellular Automata

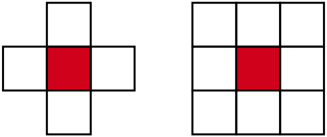
- A two-dimensional cellular automaton (CA2) is a triple (Σ, f, N) where N is a neighborhood vector.

Two-dimensional Cellular Automata



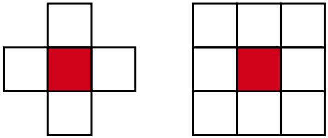
- A two-dimensional cellular automaton (CA2) is a triple (Σ, f, N) where N is a neighborhood vector.
- Most common neighborhood vector: von Neumann (left) and Moore (right).

Two-dimensional Cellular Automata



- A two-dimensional cellular automaton (CA2) is a triple (Σ, f, N) where N is a neighborhood vector.
- Most common neighborhood vector: von Neumann (left) and Moore (right).
- Its configurations are functions $s : \mathbb{Z} \to \mathbb{Z} \to \Sigma$.

Two-dimensional Cellular Automata



- A two-dimensional cellular automaton (CA2) is a triple (Σ, f, N) where N is a neighborhood vector.
- Most common neighborhood vector: von Neumann (left) and Moore (right).
- Its configurations are functions $s : \mathbb{Z} \to \mathbb{Z} \to \Sigma$.
- Similar to CA1s, there are quiescent letters and spatially-finite configurations.

Weakly-reversible CA2, Mechanized

Weakly-reversible CA2, Mechanized

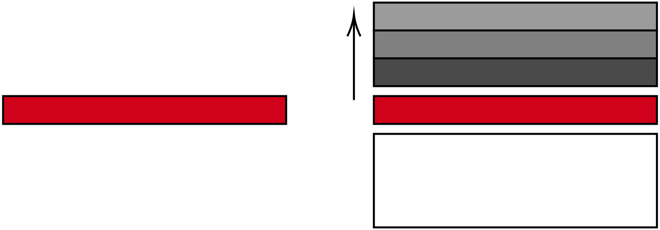
- We mechanize von Neumann CA2s with neighborhood radius 1 where cells contain $\mathcal{O}(\Sigma)$ instead of Σ.

Weakly-reversible CA2, Mechanized

- We mechanize von Neumann CA2s with neighborhood radius 1 where cells contain $\mathcal{O}(\Sigma)$ instead of Σ.
- Reduction from CA1 is done by storing the history in the additional dimension; the idea goes back to Toffoli in (Toffoli, 1977).

Weakly-reversible CA2, Mechanized

- We mechanize von Neumann CA2s with neighborhood radius 1 where cells contain $\mathcal{O}(\Sigma)$ instead of Σ.
- Reduction from CA1 is done by storing the history in the additional dimension; the idea goes back to Toffoli in (Toffoli, 1977).



Outline

Lessons Learned

Lessons Learned

Reversibility is about storing history, which requires a certain degree of control flow management.

Lessons Learned

Reversibility is about storing history, which requires a certain degree of control flow management.

- It is very hard (if not impossible) to store history in FRACTRAN.

Lessons Learned

Reversibility is about storing history, which requires a certain degree of control flow management.

- It is very hard (if not impossible) to store history in FRACTRAN.
- MM has a very restrictive control flow mechanism.

Lessons Learned

Reversibility is about storing history, which requires a certain degree of control flow management.

- It is very hard (if not impossible) to store history in FRACTRAN.
- MM has a very restrictive control flow mechanism.
- MMA/CM has a more flexible control flow mechanism but it is still not enough.

Lessons Learned

Reversibility is about storing history, which requires a certain degree of control flow management.

- It is very hard (if not impossible) to store history in FRACTRAN.
- MM has a very restrictive control flow mechanism.
- MMA/CM has a more flexible control flow mechanism but it is still not enough.
- Morita's counter machine has a flexible enough control flow mechanism.

Lessons Learned

Reversibility is about storing history, which requires a certain degree of control flow management.

- It is very hard (if not impossible) to store history in FRACTRAN.
- MM has a very restrictive control flow mechanism.
- MMA/CM has a more flexible control flow mechanism but it is still not enough.
- Morita's counter machine has a flexible enough control flow mechanism.
- Cellular automata can have almost arbitrary control flow mechanisms.

Challenges Faced

Challenges Faced

- Morita's construction involves creating a lot of fresh variables.

Challenges Faced

- Morita's construction involves creating a lot of fresh variables. We found that using a pairing function results in a more elegant mechanization.

Challenges Faced

- Morita's construction involves creating a lot of fresh variables. We found that using a pairing function results in a more elegant mechanization.
- Morita's counter machines, viewed as lists, do not provide enough structure to implement Morita's construction.

Challenges Faced

- Morita's construction involves creating a lot of fresh variables. We found that using a pairing function results in a more elegant mechanization.
- Morita's counter machines, viewed as lists, do not provide enough structure to implement Morita's construction. Our graph representation significantly simplifies our mechanization of Morita's construction.

Challenges Faced

- Morita's construction involves creating a lot of fresh variables. We found that using a pairing function results in a more elegant mechanization.
- Morita's counter machines, viewed as lists, do not provide enough structure to implement Morita's construction. Our graph representation significantly simplifies our mechanization of Morita's construction.
- The old version of binary Turing machine in the library contains too many edge cases.

Challenges Faced

- Morita's construction involves creating a lot of fresh variables. We found that using a pairing function results in a more elegant mechanization.
- Morita's counter machines, viewed as lists, do not provide enough structure to implement Morita's construction. Our graph representation significantly simplifies our mechanization of Morita's construction.
- The old version of binary Turing machine in the library contains too many edge cases. The new version of binary Turing machine[2] in the library was partly influenced by our discussion on reduction to CA1.

[2] https://github.com/uds-psl/coq-library-undecidability/pull/143

Our Contributions

Our Contributions

As far as we are aware, we are the first to mechanize the following in Coq:

Our Contributions

As far as we are aware, we are the first to mechanize the following in Coq:

- The fact that reversible FRACTRAN programs have decidable halting problems,

Our Contributions

As far as we are aware, we are the first to mechanize the following in Coq:

- The fact that reversible FRACTRAN programs have decidable halting problems,
- Partial Morita's construction: deterministic 2-Morita's counter machine to reversible and deterministic 4-Morita's counter machine,

Our Contributions

As far as we are aware, we are the first to mechanize the following in Coq:

- The fact that reversible FRACTRAN programs have decidable halting problems,
- Partial Morita's construction: deterministic 2-Morita's counter machine to reversible and deterministic 4-Morita's counter machine,
- One-dimensional and two-dimensional cellular automata, and

Our Contributions

As far as we are aware, we are the first to mechanize the following in Coq:

- The fact that reversible FRACTRAN programs have decidable halting problems,
- Partial Morita's construction: deterministic 2-Morita's counter machine to reversible and deterministic 4-Morita's counter machine,
- One-dimensional and two-dimensional cellular automata, and
- Reduction from CA1 to weakly-reversible CA2.

Thank you for your attention!

Domain overlap

Domain overlap

- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.

Domain overlap

- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in domain iff $i_1 = i_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.

Domain overlap

- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in domain iff $i_1 = i_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- A Morita's counter machine is intensionally deterministic iff none of its instructions overlap in domain.

Domain overlap

- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in range iff $j_1 = j_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- *Distinct* instructions (p_1, x_1, i_1, j_1) and (p_2, x_2, i_2, j_2) overlap in domain iff $i_1 = i_2$ and either $x_1 \neq x_2$ or $p_1 = p_2$ or either p_1 or p_2 are increment, decrement, or unconditional jump operations.
- A Morita's counter machine is intensionally deterministic iff none of its instructions overlap in domain.
- Intensional determinism is also sound.

Deterministic simulation

Let $\Rightarrow_1$ and $\Rightarrow_2$ be step relations. Assuming the following hold:

- For all configurations s_1, s_2, and t_1, if $s_1 \Rightarrow t_1$ and sync s_1 s_2 then there exists t_2 such that $s_2 \Rightarrow_2^+ t_2$ and sync t_1 t_2.
- For all configurations s_1 and s_2, if $\Rightarrow_1$ is stuck at s_1 and sync s_1 s_2 then $\Rightarrow_2$ terminates starting from s_2.
- $\Rightarrow_1$ is decidable.
- $\Rightarrow_2$ is deterministic.

then we have that for all configurations s_1 and s_2 that are in sync, $\Rightarrow_1$ terminates starting from s_1 iff $\Rightarrow_2$ terminates starting from s_2.