SAARLAND UNIVERSITY
Faculty of Mathematics and Computer Science

MASTER'S THESIS

Mechanized Undecidability of Halting Problems for Reversible Machines

Author
Hizbullah Abdul Aziz JABBAR

Advisor
Dr. Andrej DUDENHEFNER

Reviewers
Prof. Dr. Gert SMOLKA
Dr. Andrej DUDENHEFNER

Submitted: July 21, 2022

Eidesstattliche Erklärung
Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath
I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung
Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent
I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 21st July, 2022

"Those who play with the devil's toy will be brought to degrees to wield his sword."

Buckminster Fuller

Abstract

A reversible machine is a machine where no information is ever lost throughout the course of its computation. Interests in reversible machines stem from Landauer's Principle which implies that a reversible machine is capable of operating at a much lower energy consumption level compared to a non-reversible one, which also opens up possibilities of significant performance improvements.

From the point of view of computability theory, many reversible machines are as expressive as their non-reversible counterparts. It has been shown that reversible Turing machines, reversible counter machines, and reversible cellular automata (up to two dimension) are universal. However, to the best of our knowledge, none of the aforementioned results have been mechanized in the Coq Proof Assistant. As such, this work aims to close the gap and mechanize in Coq the (un)-decidability results of reversible machines, namely reversible FRACTRAN programs, reversible counter machines, and reversible cellular automata using a synthetic approach.

Acknowledgements

I am extremely grateful to my advisor Andrej for his invaluable guidance, patience, and advice throughout this project. I am humbled to have worked with him and I have definitely grown during the course of our interactions, not only as a Coq programmer but also as a student in general. I would also like to thank Prof. Smolka for his lectures and contagious passion for constructive mathematics, which is a big reason why I ended up choosing to work on this project.

I would also like to thank my friends Sunny, Waqas, Taqi, Abdallah, Masud, Zayat, Gad, Chris, Roberto, and more for their their support during my studies. I often say that Saarbrücken the city is nothing special, but the people whom I met there are exceptional.

Last but not least, my immense gratitude to both of my parents, without whom I cannot be who I am today.

Contents

Chapter 1

Introduction

1.1 Reversible Machines

A reversible machine is a machine whose computation can be traced back in time [16]. As such, no information is ever lost. If one considers machines as step relations between states of computations (often called *configurations*), then reversible machines are exactly those whose step relations are left-unique. Viewed this way, reversibility is a dual notion to determinism.

Interests in reversible machines stem from Landauer's Principle [17], stating that a decrease in entropy in the Information Bearing Degrees of Freedom (IBDF) must be accompanied by an increase in the Non-Information-Bearing Degrees of Freedom (NIBDF). For example, one may consider the voltage levels of the transistors inside a CPU as its IBDF whereas the temperature of its wirings as one of its NIBDFs. When a bit of information is deleted, the entropy of the IBDF decreases because there is less information contained within. By Landauer's Principle, the entropy of the NIBDFs must increase by the same amount via e.g. temperature increase in the wirings. If, on the other hand, the operations performed on the informations inside the IBDF are reversible, then by the same principle the CPU would avoid emitting the heat to its wirings, thus saving energy and potentially allowing for better performance [12]. See Bennett [4] for a more thorough treatment of Landauer's Principle.

From the point of view of computability theory, many reversible machines retain the expressive power of their non-reversible counterparts. For example, the (computational) universality of reversible Turing machines [3] [23], reversible counter machines [21], reversible logic gates [13], reversible one-dimensional cellular automata [22], and reversible two-dimensional cellular automata [26] have been shown. However, at the time of writing, we are not aware of any other mechanization efforts in Coq on those results.

1.2 Outline

We begin by recapitulating basic definitions in Chapter 2 including a brief introduction to synthetic undecidability, which is the mechanization approach that we use.

In Chapter 3, we showed and mechanized the fact that reversible FRACTRAN [5] is not universal, thereby providing an example of a computation model whose reversible variant is strictly less expressive. Our results here relies on various number-theoretic facts already mechanized in the Coq library of undecidability proofs [11].

We then present our partial mechanization of Morita's construction [21] in Chapter 4, showing that any deterministic two-counter machine can be simulated by a reversible and deterministic four-counter machine. Here we propose a mechanization framework that arguably simplifies our mechanization of Morita's construction and allows us to composably reason about reversibility. Later on, we argue that the last step from a reversible and deterministic four-counter machine to a reversible and deterministic two-counter machine—a compression algorithm that preserves reversibility—can be implemented on top of our framework with minimal adjustments.

Finally, in Chapter 5 we mechanize one-dimensional and two-dimensional cellular automata and the computational universality of both one-dimensional cellular automata and reversible two-dimensional cellular automata, under a weaker notion of reversibility. To the best of our knowledge, we are the first to mechanize one-dimensional and two-dimensional cellular automata in Coq.

1.3 Contributions

This thesis includes the following contributions:

- Full mechanization of the decidability of the halting problem for reversible FRACTRAN (Theorem 3.20) which relies on various number-theoretic facts already mechanized in the Coq library of undecidability proofs [11].
- Partial mechanization of Morita's construction [21] (sans the compression step). Our mechanization approach based on Morita graph (Section 4.4) arguably simplifies the overall mechanization effort while at the same time allows for composable reasoning of reversibility.
- To the best of our knowledge, the first Coq mechanization of one-dimensional and two-dimensional cellular automata. Additionally, we showed and mechanized the universality of one-dimensional cellular automata and reversible two-dimensional cellular automata, under a weaker notion of reversibility.

Many important facts and lemmas in the online version of this thesis is hyperlinked with an accompanying documentation of our Coq development. Nevertheless, readers are encouraged to also see the overall aforementioned Coq documentation at

https://www.ps.uni-saarland.de/~jabbar/master/coqdoc/.

Chapter 2

Preliminaries

2.1 Basic Types and Definitions

We work inside a constructive type theory as implemented in the Coq Proof Assistant (henceforth, Coq's type theory) [25], with a cumulative hierarchical universe of types $\mathcal{T}_i$ for natural numbers i and an impredicative universe of proposition $\mathbb{P} \subseteq \mathcal{T}$. Throughout this thesis, we use the following (inductive) data types and definitions most extensively.

Definition 2.1 (Product type and projections) *Given types X and Y, one can construct a product of X and Y, written as $X \times Y$. If $x : X$ and $y : Y$, then $(x, y) : X \times Y$. Furthermore, if $a : X \times Y$ then $a|_1 : X$ and $b|_2 : Y$ are called the first and second projection of a, respectively.*

Definition 2.2 (Sum types and injections) *Given types X and Y, one can construct a disjoint union of X and Y, written as $X + Y$. Constructing an element of a $X + Y$ can be done using injections: if $x : X$ then $\mathsf{inl}\ x : X + Y$ (left injection) and if $y : Y$ then $\mathsf{inr}\ y : X + Y$ (right injection).*

Definition 2.3 (Boolean) *A boolean $b : \mathbb{B}$ is either tt or ff, denoting the boolean truth and falsity, respectively.*

Definition 2.4 (Decidable equality) *We say that a type X has a decidable equality iff there is a function $f : X \to X \to \mathbb{B}$ such that for all $x, y : X$ we have $x =_{\mathbb{P}} y$ iff $f\ x\ y =_{\mathbb{B}} \mathsf{tt}$.*

Remark 2.5 *We use $=$ to denote both boolean and propositional equality when it is clear from the context.*

Definition 2.6 (Natural numbers) *A natural number $n : \mathbb{N}$ is either 0 or a successor of a natural number n', written as $S\ n'$.*

Fact 2.7 *For all $n, m \in \mathbb{N}$, $n = m$ is decidable.*

Proof One can use a boolean equality decider $f : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ as follows.

$$\begin{aligned} f\ 0\ 0 &= \mathsf{tt} \\ f\ 0\ (S\ m) &= \mathsf{ff} \\ f\ (S\ n)\ 0 &= \mathsf{ff} \\ f\ (S\ n)\ (S\ m) &= f\ n\ m \end{aligned}$$

Definition 2.8 (Option) *An option type $\mathcal{O}\ X$ of a type X is either a $\emptyset$ or $^\circ x$ for some $x : X$.*

Given a function $f : X \to \mathcal{O}\ Y$, we denote by $f^\uparrow$ the application of the monadic bind operator to f, defined below. Note that $f^\uparrow : \mathcal{O}\ X \to \mathcal{O}\ Y$.

Definition 2.9 (Monadic bind, option type)

$$\begin{aligned} f^\uparrow\ \emptyset &= \emptyset \\ f^\uparrow\ (^\circ x) &= f\ x \end{aligned}$$

Definition 2.10 (List) *A list type $\mathcal{L}\ X$ of a type X is either an empty list $[]$ or an element $x : X$ prepended in front of another list $L : \mathcal{L}\ X$, written as $x :: L$. We write $[a; b; c]$ as a notation for $a :: b :: c :: []$.*

The following standard functions over list are used throughout the thesis (especially in Chapter 4) and we include them here for the sake of self-containment.

Definition 2.11 ($+\!\!+$) *Let L_1 and L_2 be lists of a type X. We define L_2 appended to L_1, written as $L_1 +\!\!+ L_2$, as follows.*

$$\begin{aligned} [] +\!\!+ L_2 &= L_2 \\ (x :: L) +\!\!+ L_2 &= x :: (L +\!\!+ L_2) \end{aligned}$$

Definition 2.12 (map) *Given a function $f : X \to Y$ and a list $L : \mathcal{L}\ X$, $\mathsf{map}\ f\ L$ is an element of $\mathcal{L}\ Y$. More precisely,*

$$\begin{aligned} \mathsf{map}\ f\ [] &= [] \\ \mathsf{map}\ f\ (x :: L) &= f\ x :: \mathsf{map}\ f\ L \end{aligned}$$

Fact 2.13 *For all f and L, if $x' \in \mathsf{map}\ f\ L$ then there exists x such that $x \in L$ and $x' = f\ x$.*

Proof Follows by induction on L.

Definition 2.14 (filter) *Let $f : X \to \mathbb{B}$ be a function from a type X to booleans and $L : \mathcal{L}\ X$ be a list.*

$$\begin{aligned} \mathit{filter}\ f\ [] &= [] \\ \mathit{filter}\ f\ (x :: L) &= \begin{cases} x :: \mathit{filter}\ f\ L & f\ x = \mathit{tt} \\ \mathit{filter}\ f\ L & f\ x = \mathit{ff} \end{cases} \end{aligned}$$

Definition 2.15 (concat) *Let $L : \mathcal{L}\ (\mathcal{L}\ X)$ be a list of list of X.*

$$\begin{aligned} \mathit{concat}\ [] &= [] \\ \mathit{concat}\ (x :: L) &= x +\!\!+ \mathit{concat}\ L \end{aligned}$$

Definition 2.16 (flat-map) *Let $L : \mathcal{L}\ X$ be a list of X and $f : X \to \mathcal{L}\ X$.*

$$\mathit{flat\text{-}map}\ f = \mathit{concat} \circ \mathit{map}\ f$$

Definition 2.17 (Iteration) *Let a $f : X \to X$ for some type X. We use $f^n(x)$ for $n \in \mathbb{N}$ and $x : X$ to denote a n-fold application of f starting from x.*

Definition 2.18 (Pairing function) *A pairing function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and its inverse $\pi^{-1} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ forms a bijection between pairs of natural numbers and natural numbers.*

2.2 Reversibility and Determinism

We speak of *extensional* reversibility and *extensional* determinism as properties of a relation $\Rightarrow$ over some types X and Y.

Definition 2.19 (Extensional reversibility) *We say that a relation $\Rightarrow : X \to Y \to \mathbb{P}$ is extensionally reversible iff for all s, t, and u, whenever $s \Rightarrow u$ and $t \Rightarrow u$ then $s = t$.*

Definition 2.20 (Extensional determinism) *We say that a relation $\Rightarrow : X \to Y \to \mathbb{P}$ is extensionally deterministic iff for all s, t, and u, whenever $s \Rightarrow t$ and $s \Rightarrow u$ then $t = u$.*

Remark 2.21 *Extensional reversibility and extensional determinism are dual.*

2.3 Synthetic Undecidability

We follow the synthetic computability approach of Forster [8] where one works with computability theory without explicit computation models. Without getting too much into the field of synthetic computability, it is sufficient for our purpose to simply assume that every function definable in Coq's type theory is computable. This leads to a very elegant definition of *many-one* reductions.

Definition 2.22 (Many-one reduction) *Let X and Y be types. We say that a problem $P : X \to \mathbb{P}$ reduces to a problem $Q : Y \to \mathbb{P}$ iff there is a function $f : X \to Y$ such that for all $x : X$, we have that $P\ x$ iff $Q\ (f\ x)$.*

Using this approach, a problem Q can be shown to be undecidable if one can construct a many-one reduction from a *seed* problem P that is assumed to be undecidable [9]. In the Coq library of undecidable proofs [11], one can use various seed problems such as the halting problem for single-tape two symbol Turing machines, the Post Correspondence Problem, and the halting problem for two counter Minsky machines.

Chapter 3

On Reversible FRACTRAN

3.1 Introduction

FRACTRAN is one of the simplest computational models that has been shown to be universal [5]. Due to its simplicity, it has been used as an intermediate reduction step in the Coq library of undecidable proofs [11]. For example, in the process of establishing a reduction from k-counter machines to two-counter machines, a k-counter machine is first compiled into a FRACTRAN program [1] and the resulting FRACTRAN program is then compiled into a two-counter machine [2]. This is of special interest to us, because constructing a reversible two-counter machine out of a two-counter machine involves adding extra counters followed by a "compression" to reduce the number of counters back to two (cf. Chapter 4). However, it turns out that the halting problem for reversible FRACTRAN programs is decidable (Theorem 3.20). Consequently, any counter machine compression algorithm implemented using FRACTRAN as an intermediate step does not preserve both reversibility and universality.

3.2 FRACTRAN

We begin by recalling the definition of a FRACTRAN program and its step relation from Conway [5]. For the step relation, we follow the presentation of Larchey-Wendling and Forster [19].

Definition 3.1 (FRACTRAN programs) *A FRACTRAN program L is a list of fractions (represented as pairs of natural numbers) $(c, d) \in \mathbb{N} \times \mathbb{N}$.*

[1] `theories/FRACTRAN/FRACTRAN/mm_fractran.v` in [11]
[2] `theories/MinskyMachines/MMA/fractran_mma.v` in [11]

Definition 3.2 (FRACTRAN step relation) *A FRACTRAN program L induces a step relation $L \vdash s \to t$ between its configurations $s, t \in \mathbb{N}$ as follows.*

$$\frac{c \cdot s = d \cdot t}{(c,d) :: L \vdash s \Rightarrow t}\ \text{FRAC-0} \qquad \frac{d \nmid c \cdot s \qquad L \vdash s \Rightarrow t}{(c,d) :: L \vdash s \Rightarrow t}\ \text{FRAC-1}$$

We write $L \vdash s \Rightarrow^n t$ for an n-fold iteration of the step relation between s and t.

Definition 3.3 (FRACTRAN stuck) *We say that a FRACTRAN program L is stuck at a configuration s iff for all configuration t, it is not the case that $L \vdash s \Rightarrow t$.*

Definition 3.4 (FRACTRAN halting) *A FRACTRAN program L halts on an input s, written as $L \vdash x \Downarrow$, iff there exists t and n such that $L \vdash s \Rightarrow^n t$ and L is stuck at t.*

Problem 3.5 (FRACTRAN halting problem) *Given a FRACTRAN program L and input s, does L halt on s?*

Fact 3.6 (Decidability of FRACTRAN step relation) *For any $s \in \mathbb{N}$ and FRACTRAN program L, it is decidable whether L can make a step from s or not. That is, either there exists t such that $L \vdash s \Rightarrow t$ or L is stuck at s.*

Proof Follows by induction on L and case analysis on $d \mid (c \cdot s)$ in the inductive case.

3.3 Decidability of Reversible FRACTRAN Halting

We are ready to show that the halting problem of reversible FRACTRAN programs is decidable. The main idea is to use the fact that any FRACTRAN program with at least two elements $(a,b) :: (c,d) :: L$ that is non-redundant (cf. Definition 3.7) is necessarily irreversible (Lemma 3.9).

Definition 3.7 (Redundant FRACTRAN) *For all $a, b, c, d \in \mathbb{N}$ and FRACTRAN program L, we say that $(a,b) :: (c,d) :: L$ is a redundant FRACTRAN program iff $\mathit{gcd}(c,d) = 1$ and $b \mid d$.*

Let $(a,b) :: (c,d) :: L$ be a redundant FRACTRAN program. The first key observation is that the fraction (c,d) will never be executed since any $s \in \mathbb{N}$ that divides d also divides b. Consequently, $(a,b) :: (c,d) :: L$ behaves the same as $(a,b) :: L$.

Lemma 3.8 *For all redundant FRACTRAN programs $(a,b) :: (c,d) :: L$, we have that for all $s, t \in \mathbb{N}$, $(a,b) :: (c,d) :: L \vdash s \Rightarrow t$ iff $(a,b) :: L \vdash s \Rightarrow t$*

Proof The proof relies on a number-theoretic fact: if $\gcd(c, d) = 1$ and $b \mid d$ then for all $s \in \mathbb{N}$, if $d \mid (c \cdot s)$ then $b \mid (c \cdot s)$. In both directions, the claim follows by inversion on the assumption.

The second key observation is the fact that a non-redundant FRACTRAN program with at least two fractions is necessarily irreversible.

Lemma 3.9 *For all $a, b, c, d \in \mathbb{N}$ and FRACTRAN program L, if $\gcd(a, b) = 1$ and $b \nmid d$ then there exists $s, t, u \in \mathbb{N}$ such that $(a, b) :: (c, d) :: L \vdash s \Rightarrow u$ and $(a, b) :: (c, d) :: L \vdash t \Rightarrow u$ but $s \neq t$.*

Proof The claim follows by picking $s = c \cdot b$, $t = a \cdot d$, and $u = a \cdot c$, while relying on the fact that for all $p, q \in \mathbb{N}$, if $\gcd(p, q) = 1$ and $p \mid (q \cdot k)$, then $p \mid k$.

Together, Lemma 3.8 and Lemma 3.9 state that any reversible FRACTRAN program is either empty or a singleton. Since an empty FRACTRAN program trivially always halts, we only need to build a decider for singleton FRACTRAN programs.

Fact 3.10 *For any $s, d \in \mathbb{N}$ and FRACTRAN program L, it is not the case that $(0, d) :: L \vdash s \Downarrow$.*

Proof Follows by case analysis on d.

Fact 3.11 *For any $s, a, b, d \in \mathbb{N}$ and FRACTRAN program L, it is not the case that $(1 + a, b) :: (0, d) :: L \vdash s \Downarrow$.*

Proof If $b \mid (1 + a)$ then we are done. Otherwise, there exists some t such that $(1 + a, b) :: (0, d) :: L \vdash s \Rightarrow t$ where it get stuck. If $b \mid ((1 + a) \cdot t)$ then we are done, otherwise we apply FRAC-1. Since $(1 + a, b) :: (0, d) : L \vdash t \Rightarrow 0$, we have the desired contradiction.

Fact 3.12 *For any $c \in \mathbb{N}$, we have that $[(1 + c, 0)] \vdash s \Downarrow$.*

Proof Follows by case analysis on s, picking some non-zero t if s is zero.

Definition 3.13 (Regular FRACTRAN) *A FRACTRAN program L is called* regular *iff none of its denominators are zero.*

Fact 3.14 *For any $s \in \mathbb{N}$ and regular FRACTRAN program L, $L \vdash 0 \Rightarrow s$ implies that $s = 0$.*

Proof Follows by induction on L with inversion on the assumption in both cases.

Corollary 3.15 *For any $s \in \mathbb{N}$ and regular FRACTRAN program L, we have that for all $n \in \mathbb{N}$, $L \vdash 0 \Rightarrow^n s$ implies that $s = 0$.*

Proof Follows by induction on n and Fact 3.14.

Corollary 3.16 *For any regular FRACTRAN program L, it is not the case that for all $n \in \mathbb{N}$, $L \vdash 0 \Downarrow$.*

Proof Follows from Corollary 3.15.

Lemma 3.17 *For any non-zero $s, c, d \in \mathbb{N}$, if $d \nmid c$ then $[(c, d)] \vdash s \Downarrow$.*

Proof The proof relies on a number-theoretic fact: if $d \nmid c$, then there is a prime factor p such that for all $s, t \in \mathbb{N}$, if $s \cdot c = t \cdot d$ then the exponent of p in t is strictly less than the exponent of p in s. The claim then follows by complete induction on the exponent of p in s and case analysis on whether $[(c, d)]$ can make a step from s or not (Fact 3.6).

Lemma 3.18 *If $L = [(c, d)]$ for some $c, d \in \mathbb{N}$ and $d \neq 0$, then the halting problem for L is decidable.*

Proof Let s be the input. We pick a decider f as follows:

$$f(s) = \begin{cases} \mathsf{ff} & s = 0, c = 0 \\ d \mid c & s > 0, c > 0 \end{cases}$$

$\rightarrow$ We proceed by case analysis on s. If $s = 0$, the claim follows from Corollary 3.16. Otherwise, we proceed case analysis on c. If $c = 0$, the claim follows from Fact 3.10. Otherwise, the claim follows by case analysis on $d \mid c$.

$\leftarrow$ We proceed by case analysis on s followed by case analysis on c, noting that the claim follows trivially when $s = 0$ or $c = 0$. Otherwise, the claim follows by case analysis on $d \mid c$ and Lemma 3.17.

Lemma 3.19 *If $L = [(c, d)]$ for some $c, d \in \mathbb{N}$, then the halting problem for L is decidable.*

Proof Let s be the input. We proceed by case analysis on d and c.

1. If $d = 0$ and $c = 0$, we pick $f(s) = \mathsf{ff}$ as the decider and the claim follows from Fact 3.10.
2. If $d = 0$ and $c = c' + 1$ for some c', we pick $f(s) = \mathsf{tt}$ as the decider and the claim follows from Fact 3.12.
3. Otherwise $d \neq 0$ and the claim follows from Lemma 3.18.

Theorem 3.20 *For any reversible FRACTRAN program L, its halting problem is decidable.*

Proof We proceed by induction on the length of L. If $L = []$, the claim follows by picking $f(s) = \mathtt{tt}$ as a decider. If $L = [(a, b)]$ for some $a, b \in \mathbb{N}$, the claim follows from Lemma 3.19. Otherwise, $L = (a, b) :: (c, d) :: L'$ for some L' and we proceed by case analysis on a.

- If $a = 0$, the claim follows from Fact 3.10.
- Otherwise, a is non-zero. Here we rely on a number-theoretic fact: one can always reduce the fraction (a, b) into (a', b') such that $\gcd(a', b') = 1$ and for all $x, y \in \mathbb{N}$, $x \cdot a = y \cdot b$ implies $x \cdot a' = y \cdot b'$. Next, we proceed by case analysis on c.
  - If $c = 0$, the claim follows from Fact 3.11.
  - Otherwise, c is non-zero, which allows us to reduce (c, d) into (c', d') such that $\gcd(c', d') = 1$ and for all $x, y \in \mathbb{N}$, $x \cdot c = y \cdot d$ implies $x \cdot c' = y \cdot c'$. If $b' \mid d'$, then by Lemma 3.8 our assumption reduces to assuming that $(a, b) :: L'$ is reversible, which allows us to apply the inductive hypothesis. Otherwise $b' \nmid d'$, which together with Lemma 3.9 contradicts the assumption that L is reversible.

3.4 Discussion and Related Work

FRACTRAN is an example of a universal model of computation whose reversible counterpart is *not* universal. Intuitively, this is because there is no way for a FRACTRAN program to store its computation history, in contrast to e.g. counter machines where one can simply add extra counters to remember which instructions were executed up to this point (cf. Chapter 4). Indeed, at each step of a FRACTRAN computation, one always start from the first fraction.

Conway [5] was the first to show the universality of FRACTRAN by showing that FRACTRAN halting can simulate counter machine halting. This was later mechanized using a synthetic approach by e.g. Larchey-Wendling and Forster [19] as part of a mechanization of the DPRM Theorem [6]. Additionally, Larchey-Wendling [18] used FRACTRAN to show that entailment in Multiplicative Sub-Exponential Linear Logic is undecidable, using the same synthetic approach. As far as we are aware, we are the first to show and mechanize the fact that reversible FRACTRAN programs have decidable halting problems [3].

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

Chapter 4

On Reversible Counter Machines

4.1 Introduction

In this chapter, we present our partial mechanization of Morita's construction [21, Section 3] to show that a deterministic two-counter machine can be simulated by a deterministic and reversible two-counter machine. Morita's construction consists of three main steps:

1. reducing indegree of adresses [21, Lemma 3.1] which is explained in Section 4.6,
2. adding extra counters to record the computation history [21, Theorem 3.1] which is explained in Section 4.7, and
3. reversibly compressing back to two counters using Gödel numbering [21, Theorem 4.1].

We fully mechanize step one and two (under reasonable assumptions) and later on we argue in Section 4.9 that Morita's reversibility-preserving compression step can be done in the framework of our current mechanization. Our mechanization framework, based on Morita graphs (Section 4.4), not only results in arguably simple mechanizations, but also allows us to composably reason about reversibility: given a number of reversible counter machines, one can trivially compose them into a reversible counter machine (Lemma 4.25).

Remark 4.1 *Morita presents another step between step* 2 *and step* 3 *above, whose purpose is to reversibly erase the computation history [21, Theorem 3.2]. However, this step is not essential with respect to reversibility.*

The first step reduces the indegrees of the addresses to two or less. The notion of indegree for addresses stems from the fact that a counter machine can be viewed as a graph where addresses are nodes and instructions are edges (we further motivate and formalize this view in Section 4.4). Our choice to work with Morita's formalization of counter machines (henceforth, Morita machines) instead

of a plethora of existing counter machine formalizations is motivated by recent decidability results (cf. Section 4.2). The reason that the indegrees are reduced to two or less is because the computation history is stored as a binary number in the second step. The final step then compresses the 4-counter Morita machine back to a 2-counter Morita machine, in a manner similar to Minsky [20, Theorem 14.1-1].

Since Morita's construction proceeds in steps, there are obligations to show that a Morita machine constructed in a step simulates the Morita machine constructed in the previous step with respect to termination. To ease this process, we use a number of simulation lemmas (Section 4.3), which given few relatively straightforward preconditions yield the desired simulation.

Each step of Morita's construction involves generating fresh quadruples with fresh addresses. This necessitates an auxillary step (Section 4.5) to transform the quadruples into a form that is more amenable to create and reason about fresh quadruples and addresses. In fact, this auxillary step will be used each time we need to create fresh quadruples.

Finally, we present our overall mechanization Morita's construction without the compression step, together with its proof of correctness in Section 4.8. We then conclude with a discussions and future work in Section 4.10.

4.2 Morita Machines

When we started our mechanization of Morita's construction, there were at least two counter machine formalizations in the Coq library of undecidable proofs [11], namely MM and MMA, which are deterministic by construction. Both counter machines have the same instruction set $\{+, -\}$ containing an increment and decrement instructions, respectively, where the latter is also used for jumps. The difference lies in the semantics of their decrement instructions: a MM jumps on 0 whereas MMA jumps on positive.

Two-counter MMs, whose instructions are a subset of Minsky's original definition [20, Table 11.1] (MM is short for "Minsky Machine"), turns out to have a decidable halting problem [7, Remark 1]. Two-counter MMAs ("Minsky Machine Alternative") turn out to be still universal, so we were hoping to use MMAs as the base of our mechanization. In fact, the existing counter machine compression algorithm mentioned in Chapter 3 works on MMA. However, not only that the aforementioned compression algorithm turns out to be not reversibility-preserving (cf. Theorem 3.20), two-counter reversible MMAs also turns out to have decidable halting problems [7, Theorem 21]. As a result, we chose to use Morita machines instead, despite the fact that they can be non-deterministic.

We begin by recapitulating the formal definitions of Morita machines [21, Definition 2.1].

Definition 4.2 (Instructions) *A Morita machine has a* 5*-element instruction set* $I = \{Z, P, 0, -, +\}$ *containing zero test, positive test, unconditional jump, decrement, and increment, respectively.*

Definition 4.3 (Quadruples) *A k-quadruple* a, b, c *is an element of* $I \times \{1, \ldots, k\} \times \mathbb{N} \times \mathbb{N}$. *Let* $a = (p, x, i, j)$ *be a quadruple. We say that a contains an instruction* p*, works on a counter* x*, located at an address* i*, and points to an address* j*. In this case, we have that* $\mathsf{at}\ a = i$ *and* $\mathsf{to}\ a = j$.

Definition 4.4 (Morita machines) *A k-counter Morita machine is a list of k-quadruples.*

Definition 4.5 (Morita configurations) *Similar to other counter machines, a k-Morita configuration is a pair* $(i, \overrightarrow{v})$ *where* $i \in \mathbb{N}$ *is the current address and* $\overrightarrow{v}$ *is a vector of k natural numbers representing the current values of the counters.*

Fact 4.6 *Equality on morita configurations is decidable.*

Proof Follows from Fact 2.7.

We chose to present the small-step semantics of Morita machines in the style similar to existing counter machine formalization in the Coq library of undecidable proofs [11] instead of Morita's original presentation [21, Definition 2.2] because it is more amenable to mechanization.

Definition 4.7 (Step relation) *A k-counter Morita machine* m *induces a step relation* $M \vdash s \Rightarrow t$ *between its configurations* s, t *as follows.*

$$\text{INC}\ \frac{v[x] = w \qquad (+, x, i, j) \in m}{m \vdash (i, v) \Rightarrow (j, v[w + 1/x])} \qquad \text{DEC}\ \frac{v[x] = 1 + w \qquad (-, x, i, j) \in m}{m \vdash (i, v) \Rightarrow (j, v[w/x])}$$

$$\text{NOP}\ \frac{(0, x, i, j) \in m}{m \vdash (i, v) \Rightarrow (j, v)}$$

$$\text{ZER}\ \frac{v[x] = 0 \qquad (Z, x, i, j) \in m}{m \vdash (i, v) \Rightarrow (j, v)} \qquad \text{POS}\ \frac{v[x] = 1 + w \qquad (P, x, i, j) \in m}{m \vdash (i, v) \Rightarrow (j, v)}$$

where $v[i]$ *denotes the i-th component of* v *and* $v[n/i]$ *denotes the update of the i-th component of* v *to* n*. Furthermore, we write* $\Rightarrow^*$ *and* $\Rightarrow^+$ *as the reflexive transitive and transitive closure of* $\Rightarrow$*, respectively.*

Fact 4.8 *For all Morita machines* m, n *and configurations* s, t*, if* $m \vdash s \Rightarrow t$ *and* $m \subseteq n$ *then* $n \vdash s \Rightarrow t$.

Proof Follows by inversion on $m \vdash s \Rightarrow t$.

Fact 4.9 *For all Morita machines m, n and configurations s, t, if $m \vdash s \Rightarrow^* t$ and $m \subseteq n$ then $n \vdash s \Rightarrow^* t$.*

Proof Follows by induction on the number of steps and Fact 4.9.

Definition 4.10 (Stuck) *We say that a k-counter Morita machine m is stuck at a configuration s iff for all configuration t, it is not the case that $m \vdash s \Rightarrow t$.*

Fact 4.11 (Propositional decidability of Morita machine step relation) *For all Morita machine m and configurations s, either there exists t such that $m \vdash s \Rightarrow t$ or m is stuck at s.*

Definition 4.12 (Morita termination) *A k-counter Morita machine m terminates starting from a configuration s, written as $m \vdash s \Downarrow$, iff there exists a configuration t such that $m \vdash s \Rightarrow^* t$ and m stuck at t.*

Problem 4.13 (Morita machine halting problem) *Given a k-counter Morita machine m and an input configuration s, does m terminate starting from s?*

As previously mentioned, given a deterministic two-counter Morita machine m, Morita's construction (Section 4.8) constructs a deterministic and reversible two-counter Morita machine m' that simulates m. As such, there is a need for a syntactic characterization of reversibility that only depends on m, which is defined using the so-called *range* overlap [21, Definition 2.3].

Definition 4.14 (Range overlap) *Let $D = \{-, 0, +\}$.*
Two distinct *quadruples $a = (p_1, x_1, i_1, j_1)$ and $b = (p_2, x_2, i_2, j_2)$ overlap in range iff*

$$j_1 = j_2 \wedge (x_1 \neq x_2 \vee p_1 = p_2 \vee p_1 \in D \vee p_2 \in D).$$

In other words, a and b overlap in range iff they point to the same address and *either they work on different counters, contain the same instruction, or at least one of their instructions are in D.*

At a first glance, for a Morita machine m to be *not* reversible there must be at least two quadruples $a, b \in m$ that point to the same address. However, not all such pairs of quadruples cause non-reversibility; for example, despite the fact that $a = (Z, \hat{1}, i_1, j)$ and $b = (P, \hat{1}, i_2, j)$ point to the same address j, work on the same counter $\hat{1}$, and contain two different instructions, a and b by themselves are still reversible. This is because $\hat{1}$ cannot contain a positive number *and* zero at the same time, which means that only one of them will be executed at a time (if at all).

Remark 4.15 *In our development, we use a slightly different presentation based on an inductive predicate to ensure that if any two quadruples a and b overlap in range, then $a \neq b$.*

Definition 4.16 (Intensional reversibility) *A k-counter Morita machine m is* intensionally *(or syntactically) reversible iff for all* distinct *quaduples $a, b \in m$, a and b do* not *overlap in range.*

It turns out that Definition 4.16 is sound but not complete; that is, any intensionally reversible Morita machine is also extensionally reversible, but there is an extensionally reversible Morita machine that is not intensionally reversible.

Theorem 4.17 (Soundness, intensional reversibility) *A k-counter machine m that is intensionally reversible is also extensionally reversible.*

Proof We proceed by case analysis on equality of s and t. If $s = t$ then we are done. Otherwise, we use the hypothesis that for all distinct quadruples $a, b \in m$, a and b do not overlap in range to derive a contradiction by showing that $a \neq b$ and a and b overlap in range, which follows from inversion on $m \vdash s \Rightarrow u$ and $m \vdash t \Rightarrow u$.

Lemma 4.18 *There is an extensionally reversible counter machine that is not intensionally reversible.*

Proof Let $m = [(Z, 1, i, i), (P, 1, i, i), (0, 1, i, i)]$ be a 1-counter Morita machine that does not change its configurations i.e. for all s and t, if $m \vdash s \Rightarrow t$ then $s = t$ (this can be shown easily by inversion). This machine is therefore extensionally reversible, but $(Z, 1, i, i)$ and $(0, 1, i, i)$ overlap in range.

4.3 Simulation Lemmas

Let $\Rightarrow_1$ and $\Rightarrow_2$ be two step relations. We consider two types of simulation lemmas:

- lockstep simulation, where each step of $\Rightarrow_1$ corresponds to one step of $\Rightarrow_2$, and
- deterministic simulation, where each step of $\Rightarrow_1$ corresponds to one or more steps of $\Rightarrow_2$, with the added condition that $\Rightarrow_2$ is deterministic.

In both cases, we use a notion of synchronicity $\mathsf{sync} : X \to Y \to \mathbb{P}$ between configurations of $\Rightarrow_1$ (with type X) and $\Rightarrow_2$ (with type Y).

Lemma 4.19 (Lockstep simulation) *Let s be a configuration of $\Rightarrow_1$ and s' be a configuration of $\Rightarrow_2$. Assuming that the following holds:*

- *for all configurations s, t, and s', if $s \Rightarrow_1 t$ and $\mathsf{sync}\ s\ s'$ then there exists t' such that $s' \Rightarrow_2 t'$ and $\mathsf{sync}\ t\ t'$, and*
- *for all configurations s', t', and s, if $s' \Rightarrow_2 t'$ and $\mathsf{sync}\ s\ s'$ then there exists t such that $s \Rightarrow_1 t$ and $\mathsf{sync}\ t\ t'$,*

then we have that $\Rightarrow_1$ terminates starting from s iff $\Rightarrow_2$ terminates starting from s' if $\mathsf{sync}\ s\ s'$.

Proof For all s, t, s', and t', we have that $s \Rightarrow_1^* t$ iff $s' \Rightarrow_2^* t'$ by induction on the number of steps. Additionally, we have that $\Rightarrow_1$ is stuck at s iff $\Rightarrow_2$ is stuck at s' by our assumptions.

Lemma 4.20 (Deterministic simulation) *Let s be a configuration of $\Rightarrow_1$ and s' be a configuration of $\Rightarrow_2$. Assuming that the following holds:*

- *for all configurations s, t, and s', if $s \Rightarrow_1 t$ and $\mathsf{sync}\ s\ s'$, then there exists t' such that $s' \Rightarrow_2^+ t'$ and $\mathsf{sync}\ t\ t'$.*
- *for all s and s', if M_1 is stuck at s and $\mathsf{sync}\ s\ s'$, then M_2 terminates starting from s',*
- *$\Rightarrow_1$ is propositionally decidable, and*
- *$\Rightarrow_2$ is deterministic,*

then we have that $\Rightarrow_1$ terminates starting from s iff $\Rightarrow_2$ terminates starting from s' if $\mathsf{sync}\ s\ s'$.

Proof The left-to-right direction follows by induction on the number of steps, whereas the right-to-left direction requires a complete induction on the number of steps.

Remark 4.21 *In our development, we chose to use a different, simpler approach [1] to show Lemma 4.20. In particular, we show both directions using induction on suitable (inductive) propositions instead of the number of steps. The left-to-right direction can be shown using induction on $\Rightarrow_1^*$. For the right-to-left direction, we use the fact that $\Rightarrow_2$ is deterministic to conclude that if $\Rightarrow_2$ terminates starting from s' then s' is accessible, i.e. it satisfies the accessibility predicate [24, Section 26.1]. The claim then follows from the induction on the accessibility predicate on s' and the fact that $\Rightarrow_1$ is propositionally decidable.*

[1] in collaboration with Andrej Dudenhefner

4.4 Morita Graphs

The notion of Morita graph is the heart of our mechanization of Morita's construction. In fact, we mechanize the main steps of Morita's construction as Morita graph transformations. As we will see later, this allows for arguably simple mechanizations since each step can be defined by using map or flat-map, which in turn admits relatively simple inductive proofs.

Simply put, a Morita graph is just a list of Morita machines where each distinct pair of Morita machines in it are disjoint.

Definition 4.22 (Disjointness) *A pair of Morita machines m and n is said to be disjoint iff it is not the case that there is $a \in m$ and $b \in n$ such that $\mathsf{to}\ a = \mathsf{to}\ b$. This notion of disjointness is lifted to a list of Morita machines: we say that a list of Morita machines M is disjoint iff each* distinct *pair $m, n \in M$ is disjoint.*

Definition 4.23 (Morita graph) *A Morita graph M is a disjoint list of Morita machines.*

It turns out that disjointness is a sufficient condition to composably reason about reversibility.

Fact 4.24 *For all Morita machines m and n, if m and N are disjoint, then no quadruples $a \in M$ and $b \in N$ overlap in range.*

Proof By Definition 4.14, a and b overlap in range if, in particular, they are pointing to the same address; this of course contradicts the disjointness assumption.

Lemma 4.25 *Let M be a Morita graph. If for each $m \in M$, m is intensionally reversible, then $\mathsf{concat}\ M$ is intensionally reversible.*

Proof Follows by structural induction on M and Fact 4.24.

Definition 4.26 (Morita graph configurations) *A Morita graph configuration is exactly a Morita machine configuration.*

Definition 4.27 (Morita graph step relation) *A Morita graph M induces a step relation between its configurations as follows.*

$$\frac{m \vdash s \Rightarrow t \qquad m \in M}{M \vdash s \Rightarrow t}\ \text{IN}$$

Fact 4.28 *For all Morita graphs M, N and configurations s, t, if $M \vdash s \Rightarrow^* t$ and $M \subseteq N$ then $N \vdash s \Rightarrow^* t$.*

Proof Follows from Fact 4.9.

Remark 4.29 *The definition of Morita graph stuckness and termination are analoguos to the Morita machine i.e. Definition 4.10 and Definition 4.12.*

Fact 4.30 (Propositional decidability of Morita graph step relation) *For all Morita graph M and configurations s, either there exists t such that $M \vdash s \Rightarrow t$ or M is stuck at s.*

Proof Follows from Fact 4.11.

Given a Morita machine m, one can construct a Morita graph M such that M simulates m with respect to termination using the to-graph function defined below.

Definition 4.31 (to-graph)

$$\begin{aligned} \textit{to-graph}\ [] &= [] \\ \textit{to-graph}\ (h :: t) &= (h :: \textit{filter}\ (\lambda q.\textit{to}\ q = \textit{to}\ h)\ t) :: \textit{to-graph}\ (\textit{filter}\ (\lambda a.\textit{to}\ a \neq \textit{to}\ h)\ t) \end{aligned}$$

Remark 4.32 *One can trivially convert a Morita graph back into a Morita machine by using concat.*

Fact 4.33 *For all Morita machine m, to-graph m is disjoint.*

Proof Follows from induction on m and the fact that for all quadruples $a \in \text{filter}\ (\lambda a.\text{to}\ a \neq \text{to}\ h)\ t$, it is not the case that $\text{to}\ h = \text{to}\ a$.

Of course, to-graph does not add nor remove quadruples; it merely rearranges them.

Fact 4.34 () *For all Morita machine m and quadruple a, if $a \in m$ then there exists m' such that $m' \in$ to-graph m and $a \in m'$.*

Proof Follows from induction on m.

In fact, to-graph also gives us *uniformity* in addition to disjointness. Even though uniformity is not necessary when dealing with reversibility, it nonetheless turns out to be necessary to show simulation in the indegree reduction step in Section 4.6 and the adding counters step in Section 4.7.

Definition 4.35 (Uniformity) *We say that a Morita machine m is uniform iff for all $a \in m$, there exists j such that $\text{to}\ a = j$. In other words, every quadruples in m points to the same address. As with disjointness, we lift this notion to a list of Morita machines: a list of Morita machines M is uniform iff for all $m \in M$, m is uniform.*

Fact 4.36 *For all Morita machine m, we have that* to-graph *m is uniform.*

Proof Follows from induction on m.

Finally, we show that for all Morita machines m, to-graph m simulates m with respect to termination. Here we use the lockstep simulation lemma (Lemma 4.19), which requires us to pick a notion of synchronicity between configurations of m and to-graph m; by Definition 4.26, equality is sufficient.

Fact 4.37 *For all m, s, and t, $m \vdash s \Rightarrow t$ iff $(\textsf{to-graph}\ m) \vdash s \Rightarrow t$.*

Proof Follows from Fact 4.34.

Lemma 4.38 (to-graph simulation) *For all Morita machine m and configurations s, $m \vdash s \Downarrow$ iff $(\textsf{to-graph}\ m) \vdash s \Downarrow$.*

Proof Follows from Lemma 4.19 using Fact 4.37.

Fact 4.39 (Preservation of determinism) *For all Morita machine m, if m is deterministic then* to-graph *m is also deterministic.*

Proof Follows from Fact 4.37.

4.5 Pairing Step

We use a pairing function π to create fresh addresses. In order to do so, we map existing addresses $i \mapsto \pi\ (i, 0)$; fresh addresses can then be created by setting the suffixes (i.e. the second component of the argument of π) to a positive number.

Definition 4.40 (transform-address) *Let* fwd $(p, x, i, j) = (p, x, \pi\ (i, 0), \pi\ (j, 0))$. *We transform the addresses by applying* fwd *to each quadruple:*

$$\textsf{transform-address}\ M = \textsf{map}\ (\textsf{map}\ \textsf{fwd})\ M.$$

Definition 4.41 (Transformed) *We say that a Morita machine m is* transformed *iff for each $a \in m$, there exists j such that* to $a = \pi\ (j, 0)$. *As usual, we also lift this notion to Morita graphs: a Morita graph M is transformed iff for each $m \in M$, m is transformed.*

Fact 4.42 *For all M,* transform-address *M is transformed.*

Proof Follows directly from Definition 4.40.

We show that this transformation preserves the structure of the input Morita graph.

Fact 4.43 (Preservation of disjointness) *If M is disjoint then* transform-address *M is also disjoint.*

Proof Follows from the injectivity of π.

Fact 4.44 (Preservation of uniformity) *If M is uniform then* transform-address *M is also uniform.*

Proof Follows from the injectivity of π.

Fact 4.45 (Preservation of length) *If for each $m \in M$, m has length at most k, then for each $m' \in$ transform-address M, m' also has length at most k.*

Proof Follows from the fact that for all m, map m has the same length as m.

Finally, we show that for all Morita graph M, transform-address M simulates M with respect to termination, in a lockstep fashion; the notion of synchronicity between configurations that we need is straightforward.

Definition 4.46 (Sync) *Configurations $(i, \overrightarrow{v})$ and $(i', \overrightarrow{v'})$ are in sync, written as* sync $(i, \overrightarrow{v})$ $(i', \overrightarrow{v})$, *iff $i' = \pi\ (i, 0)$ and $\overrightarrow{v'} = \overrightarrow{v}$.*

Fact 4.47 *For all M, s, t, and s', if $M \vdash s \Rightarrow t$ and* sync *$s\ s'$, then there exists t' such that (*transform-address *$M) \vdash s' \Rightarrow t'$ and* sync *$t\ t'$.*

Proof Follows from inversion on $M \vdash s \Rightarrow t$.

Fact 4.48 *For all M, s', t', and s, if (*transform-address *$M) \vdash s' \Rightarrow t'$ and* sync *$s\ s'$, then there exists t such that $M \vdash s \Rightarrow t$ and* sync *$t\ t'$.*

Proof Follows from inversion on $M \vdash s \Rightarrow t$ and the injectivity of π.

Lemma 4.49 (transform-address simulation) *For all Morita graph M and configuration $(i, \overrightarrow{v})$, $M \vdash (i, \overrightarrow{v}) \Downarrow$ iff (*transform-address *$M) \vdash (\pi\ (i, 0), \overrightarrow{v})$.*

Proof Follows from Lemma 4.19 using Fact 4.47 and Fact 4.48.

Fact 4.50 (Preservation of determinism) *If M is deterministic then* transform-address *M is also deterministic.*

Proof Follows from Fact 4.48.

4.6 Reducing Indegree

As mentioned previously, we reduce the indegree of each addresses to two or less because the computation history will be stored as a binary number. It turns out that a uniform Morita graph M provides us with a good structure to do this, since each $m \in M$ contains exactly the quadruples that point to an address. In other words, reducing the indegree of an address to two or less amounts to reducing the length of its corresponding list to two or less.

First, we need to define a number of auxillary functions.

Definition 4.51 (GOTO) *We define an unconditional jump GOTO from address i to address j via the first counter $\hat{1}$ as follows:*

$$\mathit{GOTO}\ i\ j = (0, \hat{1}, i, j)$$

Remark 4.52 *The choice of counter does not matter since the instruction 0 is really unconditional.*

Definition 4.53 (ch-addr-to) *Let $a = (p, x, i, j)$.*

$$\mathit{ch\text{-}addr\text{-}to}\ a\ j' = (p, x, i, j')$$

Definition 4.54 (reduce-indegree-aux) *reduce-indegree-aux takes a uniform Morita machine m and returns a Morita graph.*

$$\begin{aligned}
\mathit{reduce\text{-}indegree\text{-}aux}\ [] &= [] \\
\mathit{reduce\text{-}indegree\text{-}aux}\ [a] &= [[a]] \\
\mathit{reduce\text{-}indegree\text{-}aux}\ [a; b] &= [[a; b]] \\
\mathit{reduce\text{-}indegree\text{-}aux}\ (a :: b :: c :: m') &= [\mathit{ch\text{-}addr\text{-}to}\ a\ j'; \mathit{ch\text{-}addr\text{-}to}\ b\ j'] :: \\
&\qquad \mathit{reduce\text{-}indegree\text{-}aux}\ (\mathit{GOTO}\ j'\ (\mathit{to}\ a) :: c :: m')
\end{aligned}$$

where $j' = \pi\ (\pi^{-1}(\mathit{to}\ p)|_1, 1 + \mathit{length}\ m')$.

Intuitively, if m contains two or fewer quadruples, reduce-indegree-aux leaves m unchanged. Otherwise, m contains three or more quadruples $a :: b :: c :: m'$, which reduce-indegree-aux,

1. takes the first two quadruples a and b and points them to a fresh address j',
2. creates a fresh quadruple GOTO j' (to a) that jumps from that fresh address to the old address that a and b were pointing to (recall that each quadruple in m points to the same address due to uniformity), and
3. recursively calls itself with the fresh quadruple together with the quadruples in m'.

This ensures that each $m'' \in$ reduce-indegree-aux m is uniform (Fact 4.58) and has length at most 2 (Fact 4.59). Furthermore, since m' gets smaller in the successive recursive calls to reduce-indegree-aux, each recursive calls will have a different fresh address j', which ensures disjointness (Lemma 4.57).

Fact 4.55 (Prefix) *For all uniform Morita machine m and quadruple a, if $m = h :: m'$ for some m' and $a \in$ concat (reduce-indegree-aux m) then $(\pi^{-1}\ (\text{to}\ a))|_1 = (\pi^{-1}\ (\text{to}\ h))|_1$. That is, each quadruple in reduce-indegree-aux m points to an address which shares a common prefix with the original quadruples in m.*

Proof Follows from induction on m and the uniformity assumption.

Fact 4.56 (Suffix) *For all transformed Morita machine m and quadruple a, if $a \in$ concat (reduce-indegree-aux m) then $(\pi^{-1}\ (\text{to}\ a))|_1 \leq$ length $m - 2$.*

Proof Follows from induction on m and injectivity of π. Note that the assumption that m is transformed is crucial.

Fact 4.57 *For all transformed Morita machine m, reduce-indegree-aux m is disjoint.*

Proof Follows from induction on m and Fact 4.56.

Fact 4.58 *For all Morita machine m, if m is uniform, then for each $m' \in$ reduce-indegree-aux m, m' is also uniform.*

Proof Follows from induction on m.

Fact 4.59 *For all Morita machine m we have that for each $m' \in$ reduce-indegree-aux m, m' has length at most 2.*

Proof Follows from induction on m.

Next, we show that for all Morita machine m, reduce-indegree-aux m simulates m with respect to termination. In this case, one step made by m corresponds to one or more steps made by reduce-indegree-aux m since fresh quadruples were involved. As before, we need to define a notion of synchronicity between configurations of m and reduce-indegree-aux m. Since m does not contain fresh quadruples, m can only make a step between configurations $(i, \overrightarrow{v})$ and $(j, \overrightarrow{w})$ where $i = \pi\ (i', 0)$ and $j = \pi\ (j', 0)$ for some i' and j'. Thus, the following definition of synchronicity suffices.

Definition 4.60 (Sync) *Configurations $(i, \overrightarrow{v})$ and $(j, \overrightarrow{w})$ are in sync, written as sync $(i, \overrightarrow{v})\ (j, \overrightarrow{w})$, iff $i = j$, $\overrightarrow{v} = \overrightarrow{w}$, and there exists j' such that $j = \pi\ (j', 0)$.*

Fact 4.61 *For all transformed and uniform Morita machine m and configurations s and t, if $m \vdash s \Rightarrow t$ then $(\textsf{reduce-indegree-aux}\ m) \vdash s \Rightarrow^+ t$.*

Proof We proceed by induction on m. If length $m \leq 2$ then the claim is immediate. Otherwise, we need to scrutinize which quadruple was executed by inversion on $m \vdash s \Rightarrow t$. If one of the first two quadruples in m was executed, then by Definition 4.54 we know that there is an additional step taken by reduce-indegree-aux m; in this case, the claim follows from transitivity of $\Rightarrow^+$, Fact 4.28, and the inductive hypothesis. Otherwise, the claim follows directly from the inductive hypothesis.

Corollary 4.62 *For all transformed and uniform Morita machine m and configurations s, t, and s', if $m \vdash s \Rightarrow t$ and $\textsf{sync}\ s\ s'$ then there exists t' such that $(\textsf{reduce-indegree-aux}\ m) \vdash s' \Rightarrow^+ t'$ and $\textsf{sync}\ t\ t'$.*

Proof Follows from Fact 4.61.

Fact 4.63 *For all transformed and uniform Morita machine m and configuration s', t', and s, if $(\textsf{reduce-indegree-aux}\ m) \vdash s' \Rightarrow t'$ and $\textsf{sync}\ s\ s'$ then there exists t such that $m \vdash s \Rightarrow t$.*

Proof Follows by inversion on $(\textsf{reduce-indegree-aux}\ m) \vdash s' \Rightarrow t'$ followed by induction on m.

Remark 4.64 *At a first glance, the statement of Fact 4.63 seems strange. We claim that one step of m corresponds to one or more steps of reduce-indegree-aux m but Fact 4.63 assumes that reduce-indegree-aux m made* one *step and concludes that m also made one step. However, as we will see, this fact is used to show that if m is stuck at some configuration s, then reduce-indegree-aux m terminates starting from a synchronized configuration s'.*

Remark 4.65 *Note that the transform-address step is crucial to be able to prove Fact 4.57, Corollary 4.62, and Fact 4.63, because it provides the exact precondition to those facts.*

Finally, we are ready to define the reduce-indegree step itself and show that for all transformed and uniform Morita graph M, reduce-indegree M simulates M with respect to termination.

Definition 4.66 (reduce-indegree)

$$\textsf{reduce-indegree} = \textsf{flat-map reduce-indegree-aux}$$

Fact 4.67 (Preservation of disjointness) *For all transformed and uniform Morita graph M, reduce-indegree M is a Morita graph.*

Proof The fact that reduce-indegree M is a list of Morita machines is immediate, so we only need to show that reduce-indegree M is disjoint. The disjointness of reduce-indegree M follows by structural induction on M and Fact 4.57.

Fact 4.68 (Preservation of uniformity) *If M is uniform then* reduce-indegree *M is also uniform.*

Proof Follows from Fact 4.58.

Lemma 4.69 *For all transformed and uniform Morita graph M and configurations s, t, and s', if $M \vdash s \Rightarrow t$ and* sync *s s' then there exists t' such that (*reduce-indegree *M) $\vdash$ $s' \Rightarrow t'$ and* sync *t t'.*

Proof Follows from Corollary 4.62 and Fact 4.28.

Lemma 4.70 *For all transformed and uniform Morita graph M and configurations s and s', if M is stuck at s and* sync *s s' then* reduce-indegree *M terminates starting from s'.*

Proof Follows by picking $t = s'$ and Fact 4.63.

Lemma 4.71 (reduce-indegree simulation) *For all deterministic, transformed, and uniform Morita graph M, an address i, and counter values $\overrightarrow{v}$, we have that $M \vdash (\pi\ (i, 0), \overrightarrow{v}) \Downarrow$ iff (*reduce-indegree *M) $\vdash (\pi\ (i, 0), \overrightarrow{v}) \Downarrow$.*

Proof Follows from Lemma 4.20 using Lemma 4.69, Lemma 4.70, and Fact 4.30, together with the assumption that reduce-indegree preserves determinism.

Remark 4.72 *We did not show the fact that* reduce-indegree *preserves determinism since it would require a comparable effort to show determinism as it does to show reversibility.*

4.7 Adding Counters

We begin by defining two of subroutines MOVE and DOUBLE. The former is a subroutine to transfer the value of a counter into another and while the latter is a subroutine to double the value of a counter.

Remark 4.73 *Morita [21, Theorem 3.1] presents MOVE and DOUBLE as one subroutine. Here (and in our development) we factor them out and present them separately for ease of mechanization and presentation. Furthermore, we are being explicit in how we create fresh quadruples with disjoint addresses using a pairing function π, which means that we have to pick the correct arguments to π.*

Definition 4.74 (MOVE) *MOVE sp ss ep es src dst transfers the value of the counter src to counter dst, with addresses starting from $\pi\ (sp, ss)$ and ending in $\pi\ (ep, es)$. It*

consists of the following quadruples:

$$(Z, dst, \pi\ (sp, ss), \pi\ (sp, 1 + ss)) \tag{4.1}$$
$$(Z, src, \pi\ (sp, 1 + ss), \pi\ (ep, es)) \tag{4.2}$$
$$(P, src, \pi\ (sp, 1 + ss), \pi\ (sp, 2 + ss)) \tag{4.3}$$
$$(-, src, \pi\ (sp, 2 + ss), \pi\ (sp, 3 + ss)) \tag{4.4}$$
$$(+, dst, \pi\ (sp, 3 + ss), \pi\ (sp, 4 + ss)) \tag{4.5}$$
$$(P, dst, \pi\ (sp, 4 + ss), \pi\ (sp, 1 + ss)) \tag{4.6}$$

Intuitively, MOVE is a loop that runs as long as src contains a positive number, decrementing src and incrementing dst at each iteration. Note that Quadruple 4.1 first checks whether dst contains 0 which means that MOVE gets stuck if dst does not contain 0.

Remark 4.75 *It may seem that the Quadruple 4.6 serves no purpose; after all, since counters contain natural numbers, an increment to a counter would ensure that said counter contains a positive number. However, it turns out that this quadruple is crucial in defining a loop without ranger overlap. Specifically, only Quadruple 4.1 and Quadruple 4.6 point to the address $\pi\ (sp, 1 + ss)$, but they do not overlap in range by Definition 4.14. Had we not use Quadruple 4.6 and change the Quadruple 4.5 to point to $\pi\ (sp, 1 + ss)$ instead, we would have a range overlap.*

Fact 4.76 *For all prefixes sp ss, suffixes ep es, and natural number n, we have that*

$$(\mathit{MOVE}\ sp\ ss\ ep\ es\ \hat{1}\ \hat{2}) \vdash (\pi\ (sp, ss), n :: 0 :: v) \Rightarrow^{+} (\pi\ (ep, es), 0 :: n :: v).$$

Proof Here we need to get rid of Quadruple 4.1 because we need to strengthen the statement so that $\hat{2}$ contains m:

$$\forall m, (\mathsf{MOVE}\ sp\ ss\ ep\ es\ \hat{1}\ \hat{2}) \vdash (\pi\ (sp, 1 + ss), n :: m :: v) \Rightarrow^{+} (\pi\ (ep, es), 0 :: n + m :: v)$$

The claim then follows by transitivity of $\Rightarrow^{+}$ and induction on n.

Definition 4.77 (DOUBLE) *DOUBLE sp ss ep es dst src doubles the value of the counter src and* simultaneously *moves it to the counter dst, with addresses starting from $\pi\ (sp, ss)$ and ending in $\pi\ (ep, es)$. In other words, if the counter src contains n, by the end of the execution of DOUBLE sp ss ep es dst src, dst will contain 2n and src will*

contain 0. *It consists of the following quadruples:*

$$(Z, src, \pi\ (sp, ss), \pi\ (ep, es)) \tag{4.7}$$
$$(P, src, \pi\ (sp, ss), \pi\ (sp, 1 + ss)) \tag{4.8}$$
$$(-, src, \pi\ (sp, 1 + ss), \pi\ (sp, 2 + ss)) \tag{4.9}$$
$$(+, dst, \pi\ (sp, 2 + ss), \pi\ (sp, 3 + ss)) \tag{4.10}$$
$$(P, dst, \pi\ (sp, 3 + ss), \pi\ (sp, 4 + ss)) \tag{4.11}$$
$$(+, dst, \pi\ (sp, 4 + ss), \pi\ (sp, 5 + ss)) \tag{4.12}$$
$$(P, dst, \pi\ (sp, 5 + ss), \pi\ (sp, ss)) \tag{4.13}$$

Intuitively, DOUBLE *is a loop that runs as long as src contains a positive number, decrementing src and incrementing dst* twice *at each iteration.*

Remark 4.78 *Note the trick with checking for positive number* after *an increment i.e. Quadruple 4.11 and Quadruple 4.13.*

Fact 4.79 *For all prefixes sp ss, suffixes ep es, and natural number n, we have that*

$$\forall m, (\mathit{DOUBLE}\ sp\ ss\ ep\ es\ \hat{1}\ \hat{2}) \vdash (\pi\ (sp, ss), m :: n :: v) \Rightarrow^{+} (\pi\ (ep, es), 2n + m :: 0 :: v).$$

Proof Follows from induction on n.

Now we are ready to put the subroutines together.

Definition 4.80 (Upcast) *Let $a = (p, x, i, j)$ be a quadruple. The upcast operator applied to a, written as $a^{\uparrow}$, is defined as*

$$(p, x, i, j)^{\uparrow} = (p, x + 2, i, j),$$

that is, the counter of a is shifted by two to the right. In other words, if e.g. a works on the counter $\hat{1}$, then $a^{\uparrow}$ works on the counter $\hat{3}$.

Definition 4.81 (add-counters-aux)

$$\begin{aligned}
\mathit{add\text{-}counters\text{-}aux}\ [a; b] =& [\mathit{ch\text{-}addr\text{-}to}\ a^{\uparrow}\ (\pi\ (j, 1))] +\!\!+ \\
& \mathit{MOVE}\ j\ 1\ j\ 11\ \hat{1}\ \hat{2} +\!\!+ \\
& [\mathit{ch\text{-}addr\text{-}to}\ b^{\uparrow}\ (\pi\ (j, 6))] +\!\!+ \\
& \mathit{MOVE}\ j\ 6\ j\ 15\ \hat{1}\ \hat{2} +\!\!+ \\
& \mathit{DOUBLE}\ j\ 11\ j\ 0 \\
\mathit{add\text{-}counters\text{-}aux}\ m =& m
\end{aligned}$$

where $j = (\pi^{-1}\ (to\ a))|_1$ i.e. the prefix of the address that a is pointing to.

Definition 4.81 deserves a more thorough explanation. First of all, the suffixes are defined such that if m contains two elements or fewer then reduce-indegree-aux m contains no quadruples that overlap in range (Lemma 4.95). Furthermore, notice that the second call to MOVE ends in suffix 15, which corresponds to Quadruple 4.12 in DOUBLE.

As mentioned previously, the computation history is stored as a binary number. Here we elected to use counter $\hat{1}$ to store the computation history and counter $\hat{2}$ as an auxillary counter, hence the need for the upcast operator.

Remark 4.82 *Given a k-counter Morita machine, Morita [21, Theorem 3.1] uses the counters $k+1$ and $k+2$ as the history counter and the auxillary counter, respectively.*

Suppose that during the course of its computation, the current value of $\hat{1}$ is n. If a was executed by, then add-counters-aux first execute a, moves n to counter $\hat{2}$, and then doubles the value of counter $\hat{2}$ and stores it in $\hat{1}$. As a result, the counter $\hat{1}$ now contains $2n$. Otherwise, b was executed, in which add-counters-aux first execute b, moves n to counter $\hat{2}$, executes Quadruple 4.12 in DOUBLE which increments the value in $\hat{1}$ to 1, and then doubles the value of counter $\hat{2}$ and stores it in $\hat{1}$. This results in counter $\hat{1}$ now containing $2n+1$. We formalize these two intuitions into the following facts.

Fact 4.83 *Let $m = [a; b]$ be a transformed Morita machine containing two quadruples a and b. If the execution of a changes the configuration $(i, \overrightarrow{v})$ to $(j, \overrightarrow{w})$ then we have*

$$(\textit{add-counters-aux } m) \vdash (i, n :: 0 :: v) \Rightarrow^{+} (j, 2n :: 0 :: v).$$

Proof Follows from transitivity of $\Rightarrow^{+}$ together with Fact 4.76 and Fact 4.79.

Fact 4.84 *Let $m = [a; b]$ be a transformed Morita machine containing two quadruples a and b. If the execution of b changes the configuration $(i, \overrightarrow{v})$ to $(j, \overrightarrow{w})$ then we have*

$$(\textit{add-counters-aux } m) \vdash (i, n :: 0 :: v) \Rightarrow^{+} (j, 2n+1 :: 0 :: v).$$

Proof Follows from transitivity of $\Rightarrow^{+}$ together with Fact 4.76 and Fact 4.79.

The add-counters step itself can be defined by a simple map.

Definition 4.85 (add-counters)

$$\textit{add-counters} = \textit{map add-counters-aux}$$

Remark 4.86 *Morita [21, Theorem 3.1] implements this step as applying add-counters-aux for each pair of quadruples that overlap in range. Had we tried to define this step on a Morita machine instead of a Morita graph, it would not be as simple as a map since the two quadruples that overlap in range can be anywhere in the list. Furthermore, Definition 4.54*

does not check whether the input pair of quadruples overlap in range or not; we merely assume uniformity. While this could result in more fresh quadruples than strictly necessary, it arguably simplifies our proofs and more importantly, it does not affect reversibility.

Next, we show that add-counters preserves disjointness, which is essential in ensuring that we can still composably reason about reversibility.

Fact 4.87 *For all transformed Morita machines m and n whose lengths are at most 2, if m and n are disjoint then add-counters-aux m and add-counters-aux n are also disjoint.*

Proof Follows by case analysis on m and n and injectivity of π, noting that the prefixes in the calls to MOVE and DOUBLE are different for m and n by our disjointness assumption.

Lemma 4.88 (Preservation of disjointness) *For all transformed Morita graph M such that for all $m \in M$, length $m \leq 2$, we have that add-counters M is also a Morita graph.*

Proof The main obligation here is to show that add-counters M is disjoint, which follows by induction on M and Fact 4.87.

As with the indegree reduction step in Section 4.6, each step made by m corresponds to one or more steps made by add-counters-aux m due to a number of fresh quadruples.

Definition 4.89 (Sync) *We say that configurations $(i, \overrightarrow{v})$ and $(j, \overrightarrow{w})$ are in sync iff $i = j$, $i = \pi\ (k, 0)$ for some k, and $\overrightarrow{w} = n :: 0 :: \overrightarrow{v}$ for some n.*

Fact 4.90 *For all transformed and uniform Morita machine m with at most two elements and configurations s, t, and s', if $m \vdash s \Rightarrow t$ and sync s s' then there exists t' such that (add-counters-aux m) $\vdash s' \Rightarrow t'$ and sync t t'.*

Proof We proceed by case analysis on m. If m is empty or a singleton, then the claim is immediate by picking $t' = t$. Otherwise, $m = [a; b]$ for some quadruples a and b. By inversion on $m \vdash s \Rightarrow t$, we have that either a or b was executed. The claim then follows from Fact 4.83 and Fact 4.84.

Lemma 4.91 *For all transformed and uniform Morita graph M such that for all $m \in M$, length $m \leq 2$, and configurations s, t, and s', if $M \vdash s \Rightarrow t$ and sync s s' then there exists t' such that (add-counters M) $\vdash s' \Rightarrow t'$ and sync t t'.*

Proof By inversion on $M \vdash s \Rightarrow t$, we have an $m' \in M$ such that $m' \vdash s \Rightarrow t$. The claim follows from case analysis on m' and Fact 4.90.

Lemma 4.92 *For all transformed Morita graph M such that for all $m \in M$, length $m \leq 2$, and configurations s and s', if M is stuck at s and sync s s' then add-counters M terminates starting from s'.*

Proof We start by picking $t = s'$. By inversion on $(\mathsf{add\text{-}counters}\ M) \vdash s' \Rightarrow t'$, we have an $m' \in \mathsf{add\text{-}counters}\ M$ such that $m' \vdash s' \Rightarrow t'$. By Fact 2.13, there is an $m \in M$ such that $m' = \mathsf{add\text{-}counters\text{-}aux}\ m$. The claim then follows by case analysis on m.

Lemma 4.93 (add-counters simulation) *For all deterministic, transformed, and uniform Morita graph M such that for each $m \in M$, $\mathsf{length}\ m \leq 2$,$\mathsf{add\text{-}counters}\ M$ simulates M with respect to termination.*

Proof Follows from Lemma 4.20 using Lemma 4.91, Lemma 4.92, and Fact 4.30, together with the assumption that add-counters preserves determinism.

Remark 4.94 *As before, we did not show the fact that reduce-indegree preserves determinism. It would require a comparable effort to show determinism as it does to show reversibility.*

Finally, we show that this step creates an intensionally reversible Morita graph.

Lemma 4.95 *For all transformed and uniform Morita machine m with at most two elements, add-counters-aux m is intensionally reversible.*

Proof Follows from case analysis on m.

Remark 4.96 *The proof of Lemma 4.95 is deceptively simple. Indeed, using a naive case analysis to show Lemma 4.95 results in a large number of cases, because if $m = [a; b]$ for some quadruples a and b then add-counters-aux m is a Morita machine with 21 quadruples. Since we have to check for every pair of quadruples, this can get unwieldy rather quickly. Instead, by noting that only a small number of quadruples in add-counters-aux m point to same addresses, we only need to consider them and not the whole machine. This requires more effort in our development than a mere case distinction even though conceptually it is.*

4.8 Morita's Construction

Finally, we are ready to formally define our partial mechanization of Morita's construction [21] specialized to two counters, that is, given a deterministic two-counter Morita machine, constructs a reversible and deterministic four-counter Morita machine.

Definition 4.97 (morita-construction)

$$\begin{aligned}\mathit{morita\text{-}construction} = {} & \mathit{concat} \circ \\ & \mathit{add\text{-}counters} \circ \mathit{transform\text{-}address} \circ \\ & \mathit{reduce\text{-}indegree} \circ \mathit{transform\text{-}address} \circ \\ & \mathit{to\text{-}graph}\end{aligned}$$

Lemma 4.98 (morita-construction simulation) *Let m be a deterministic two-counter Morita machine and $(i, \overrightarrow{v})$ a two-counter Morita machine configuration. Assuming that reduce-indegree and add-counters preserve determinism, we have that $m \vdash (i, v) \Downarrow$ iff $(\textit{morita-construction}\ m) \vdash (\pi\ (\pi\ (i, 0), 0), 0 :: 0 :: \overrightarrow{v})$.*

Proof Follows from Lemma 4.38, Lemma 4.49, Lemma 4.71, and Lemma 4.93.

Theorem 4.99 *Deterministic two-counter Morita machine halting reduces to deterministic and reversible four-counter Morita machine halting.*

Proof Let m be a Morita machine and $(i, \overrightarrow{v})$ be a two-counter Morita machine configuration. We use morita-construction to transform the input Morita machine m and we transform the input configuration $(i, \overrightarrow{v})$ into $(\pi\ (\pi\ (i, 0)), \overrightarrow{v})$. The claim follows from Lemma 4.98.

4.9 Compression

In this section, we briefly explain the compression algorithm used by Morita [21, Theorem 4.1] adapted to our Morita graph framework and argue that it is indeed reversibility-preserving. The idea behind the compression algorithm itself, namely using Goedel numbering, is not new and goes back to at least Minsky [20]; the fact that it is universality-preserving is well-understood and thus we will focus on its reversibility-preserving aspect.

Specifically, suppose that M is a Morita graph whose counter values are $[v_1; v_2; v_3; v_4]$. One can then pack the four counter values into one counter value via prime exponentiation, that is, one would work with $[2^{v_1}3^{v_2}5^{v_3}7^{v_4}; 0]$ instead (the counter $\hat{2}$ is auxillary). Consequently, the operations of M must be adapted e.g. increments becomes multiplications and decrements becomes divisions. A multiplication can be done using a loop similar to MOVE and DOUBLE; in fact, MOVE followed by DOUBLE is a subroutine to multiply by 2. Definition 4.100 provides us with a way to divide by 2.

Definition 4.100 (HALVE) *Let $[v_1'; v_2]$ be the value of the two counters. Dividing the v_1' by 2 can be done via the following.*

1. *Transfer the value of v_1' to counter $\hat{2}$ using MOVE (Definition 4.74). Recall that MOVE first checks whether $\hat{2}$ is zero or not.*

2. *Use the following quadruples to halve the value of the counter* $\hat{2}$ *and put it into the counter* $\hat{1}$*:*

$$(Z, \hat{1}, \pi\ (sp, ss), \pi\ (ep, es)) \tag{4.14}$$
$$(P, \hat{1}, \pi\ (sp, ss), \pi\ (sp, 1 + ss)) \tag{4.15}$$
$$(-, \hat{1}, \pi\ (sp, 1 + ss), \pi\ (sp, 2 + ss)) \tag{4.16}$$
$$(-, \hat{1}, \pi\ (sp, 2 + ss), \pi\ (sp, 3 + ss)) \tag{4.17}$$
$$(+, \hat{2}, \pi\ (sp, 3 + ss), \pi\ (sp, 4 + ss)) \tag{4.18}$$
$$(P, \hat{2}, \pi\ (sp, 4 + ss), \pi\ (sp, ss)) \tag{4.19}$$

Division by 3, 5, *and* 7 *are analoguous i.e. instead of* 2 *decrements between Quadruple 4.19 and Quadruple 4.18, one needs* 3, 5, *and* 7 *decrements, respectively, with the appropriate adjustments on the suffixes.*

Remark 4.101 *Note that if the counter* $\hat{2}$ *is not divisible by* 2 *then HALVE would get stuck. This is not a problem in terms of simulation, since this is equivalent to the original Morita graph trying to decrement a counter that contains zero; in which case it would get stuck as well.*

Note the trick with checking for positive number after an increment in Definition 4.100 in Quadruple 4.18 and Quadruple 4.19. As previously mentioned in Remark 4.75, this is the key insight in how one could construct a reversible loop. Furthermore, since compression algorithm amounts to replacing a quadruple with one or more quadruples e.g. a decrement would be replaced by HALVE, the overall procedure compress can be implemented using a flat-map over the input Morita graph.

4.10 Discussion and Related Work

An important lesson from Morita's construction [21] is how one can construct a reversible loop using a seemingly redundant quadruple (Remark 4.75), which as we have seen in e.g. Section 4.7, is crucial in showing the universality of reversible Morita machines.

Another important aspect of Morita's construction is the notion of intensional reversibility (Definition 4.16) based on the notion of range overlap [21, Definition 2.3]. This syntactic notion is more convenient to work with for our mechanizations, but first we had to make sure that it is a correct notion to use. Morita did not explicitly show that intensional reversibility implies extensional reversibility even though extensional reversibility is closer to the original notion of reversibility in Physics. Nevertheless, we showed that intensional reversibility is indeed sound (Theorem 4.17) but incomplete (Lemma 4.18).

Compared to the existing counter machine mechanizations in the Coq library of undecidable proofs [11], namely MM/MMA, Morita machines have a richer instruction set and admit non-determinism. This results in reversible two-counter Morita machines being more expressive than their MMA counterparts since the latter has been shown to be non-universal [7, Theorem 21]. Non-determinism is of course a downside, however, a reduction from MMA to Morita machines that preserves determinism is straightforward.

As far as we are aware, we are the first to mechanize reversible counter machines in Coq. There are quite a number of mechanizations involving counter machines in the Coq library of undecidable proofs [11] that use MMA as an intermediate problem [18] [19] [10].

Chapter 5

On Reversible Cellular Automata

5.1 Introduction

In this chapter we show that any one-dimensional cellular automaton can be simulated by a two-dimensional reversible cellular automaton, under a weaker notion of extensional reversibility where one does not consider halting configurations. Furthermore, we establish the undecidability of the halting problem of weakly-reversible, two-dimensional cellular automata via a reduction chain starting from the halting problem of binary Turing Machines.

This chapter is organized as follows. We start by recalling the definition of binary Turing machines and one-dimensional cellular automata together with suitable definitions of their halting problems. We then show that cellular automata can simulate binary Turing machines, thus establishing the fact that the halting problem of binary Turing machines reduces to the halting problem of one-dimensional cellular automata. This is the first part of the reduction chain. Finally, we complete the reduction chain by formalizing weakly-reversible (c.f. Definition 5.53) two-dimensional cellular automata and showing that weakly-reversible two-dimensional cellular automata simulate one-dimensional cellular automata.

5.2 Binary Turing Machine

We begin by recalling the definition of a simple binary Turing machine (SBTM) from the Coq library of undecidable proofs [11]. Due to its simplicity, SBTM is an excellent seed problem.

Definition 5.1 (Binary Turing machine) *A binary Turing Machine $M = (Q, \delta)$ where Q is a finite set of states and $\delta : Q \times \mathbb{B} \to \mathcal{O}\,(Q \times \mathbb{B} \times \{\mathsf{left}, \mathsf{right}\})$ is a finite transition table.*

Definition 5.2 (Binary Turing machine configurations) *An SBTM operates on a Boolean-valued tape $(l, b, r) : \mathcal{L}\ \mathbb{B} \times \mathbb{B} \times \mathcal{L}\ \mathbb{B}$, whose configurations are pairs of its states $q \in Q$ and the values of its tape.*

Like all Turing machines, an SBTM M operates by moving its head and writes Boolean values on its tape.

Definition 5.3 (Move) *We define a function θ that moves the head of a binary Turing Machine along its tape $t = (l, b, r)$ one step towards direction $d \in \{\mathsf{left}, \mathsf{right}\}$ as follows:*

$$\begin{aligned}
\theta_{\mathsf{left}}([], b, r) &= ([], \mathit{ff}, b :: r) \\
\theta_{\mathsf{left}}(h :: l, b, r) &= (l, h, b :: r) \\
\theta_{\mathsf{right}}(l, b, []) &= (b :: l, \mathit{ff}, []) \\
\theta_{\mathsf{right}}(l, b, h :: r) &= (b :: l, h, r)
\end{aligned}$$

Definition 5.4 (Step) *A binary Turing Machine $M = (Q, \delta)$ updates its configurations using a function S defined as follows:*

$$S(q, (l, b, r)) = \begin{cases} \emptyset & \delta(q, b) = \emptyset \\ {}^{\circ}(q', \theta\ d\ (l, b', r)) & \delta(q, b) = {}^{\circ}(q', b', d) \end{cases}$$

Finally, we define the SBTM halting problem as follows.

Definition 5.5 (SBTM halting) *A binary Turing Machine halts on an input c iff there exists n such that $(S^{\uparrow})^n({}^{\circ}c) = \emptyset$.*

Fact 5.6 *It is decidable whether an SBTM configuration c is a halting configuration or not.*

Proof The claim follows by evaluating $\delta(c)$.

Problem 5.7 (SBTM halting problem) *Given an SBTM M and a configuration c, does M halts on c?*

Note that due to the use of monadic bind (c.f. Definition 2.9), an SBTM technically can still make a step even though it is already in a halting configuration. In other words, if an SBTM $M = (Q, \delta)$ does halt on an input configuration c, there are many n such that $(S^{\uparrow})^n({}^{\circ}c)$. However, we are often interested in the least of such n. This motivates the following definition of the SBTM halting problem.

Definition 5.8 (SBTM minimal halting) *A binary Turing Machine halts on an input c iff there exists n such that $(S^{\uparrow})^n({}^{\circ}c) = \emptyset$ and for all $m < n$, $(S^{\uparrow})^n({}^{\circ}c) \neq \emptyset$.*

Problem 5.9 (SBTM minimal halting problem) *Given an SBTM M and a configuration c, does M minimally halts on c?*

Fact 5.10 *Definition 5.5 and Definition 5.8 are equivalent.*

Proof Definition 5.8 clearly implies Definition 5.5. The other direction follows from [24, Corollary 17.3.3] and Fact 5.6.

5.3 One-dimensional Cellular Automata

We begin by recapitulating the definition of a one-dimensional cellular automata (CA1) from the literature.

Definition 5.11 (CA1 [16]) *A CA1 is a triple (Σ, r, f) where Σ is a finite and discrete set of alphabet symbols (sometimes called* states *in the literature), r is the neighborhood radius, and $f : \Sigma^{2r+1} \to \Sigma$ is the* local update function. *A CA1 configuration is a function $\mathbb{Z} \to \Sigma$ comprising of cells corresponding to each integers. A CA1 updates its configurations c using its* global update function *G by simultaneous application of f,*

$$G(c)(i) = f(c(i-r), c(i-r+1), \ldots, c(i+r-1), c(i+r)) \tag{5.1}$$

for all $i \in \mathbb{Z}$.

For the remainder of this chapter, we assume that $r = 1$.

We mechanize CA1 configurations as triples (l, m, r) where $l, r : \mathbb{N} \to \mathcal{O}(\Sigma)$ and $m : \mathcal{O}(\Sigma)$. We split the integers in this way because $\mathbb{N}$ is simpler to work with in Coq. While it is clear that there is a bijection from $\mathbb{N} + () + \mathbb{N}$ to $\mathbb{Z}$, the choice of using $\mathcal{O}(\Sigma)$ instead of Σ is motivated by the need to model halting:

Definition 5.12 (CA1 halting configuration) *A CA1 configuration (l, m, r) is a halting configuration iff*

$$m = \emptyset \vee \exists n.l(n) = \emptyset \vee \exists n.r(n) = \emptyset$$

Consequently, the local update function f now returns $\mathcal{O}(\Sigma)$ instead of Σ.

We define a CA1 global transition function G_f with respect to a CA1 local update function f as follows,

Definition 5.13 (CA1 global transition function)

$$G_f(l, m, r) = (l', m', r')$$

where

$$m' = f^{\uparrow}(l(0), m, r(0))$$

$$l'(n) = \begin{cases} f^{\uparrow}(l(1), l(0), m) & n = 0 \\ f^{\uparrow}(l(n+1), l(n), l(n-1)) & n > 0 \end{cases}$$

$$r'(n) = \begin{cases} f^{\uparrow}(m, r(0), r(1)) & n = 0 \\ f^{\uparrow}(r(n-1), r(n), r(n+1)) & n > 0 \end{cases}$$

Note that G_f is deterministic when f is.

Fact 5.14 *For all configuration c, if c is a halting configuration then $G_f(c)$ is also a halting configuration.*

Corollary 5.15 *For all $n \in \mathbb{N}$ and configuration c, if c is a halting configuration then $G_f^n(c)$ is also a halting configuration.*

Next, we define a notion of finite configuration for CA1, which requires a notion of quiescent states. Since we split the integers and modify the local update function to take output optional values, we need to modify the definition of quiescent states and finite configurations [15, Subsection 2.3] as follows.

Definition 5.16 (Quiescent state) *A state $q \in \Sigma$ is called a* quiescent *state iff*

$$f(q, q, q) = {}^{\circ}q$$

Remark 5.17 *A configuration containing only a quiescent state q is called a quiescent configuration [15, Subsection 2.3]. Quiescent configurations have been used in the literature to model termination of cellular automata. Specifically, one can say that a cellular automaton halts if it reaches a quiescent configuration [2]. However, since self-loop is trivially non-reversible and quiescent states induce self-loop (cf. Definition 5.16), we chose to model halting* without *using quiescent states but by using $\emptyset$ instead (Definition 5.12).*

Definition 5.18 (Finite configuration) *A configuration (l, m, r) is called (spatially)* finite *with respect to a quiescent state q iff*

$$\exists n. \forall i >= n. l(i) = q \wedge r(i) = q.$$

Finally, we define the halting problem for CA1 as follows:

Definition 5.19 (CA1 minimal halting) *A CA1 halts on a* finite *configuration c iff there exists $n \in \mathbb{N}$ such that $G_f^n(c)$ is halting and $\forall m < n. G_f^m(c)$ is not halting.*

Problem 5.20 (CA1 minimal halting problem) *Given a CA1 C and a finite configuration c, does C halts starting from c?*

Remark 5.21 *We require the input configuration c to be finite to ensure that the CA1 is not too powerful. Without this requirement, one could construct a CA1 that simulates a Turing machine in only one step, for example.*

Remark 5.22 *One could formulate a simpler halting problem definition of CA1 by dropping the second conjunct in Problem 5.20, similar to Problem 5.7 for SBTM. However, since it is undecidable whether a CA1 configuration is halting or not, the two definitions of the halting problem for CA1 are not equivalent. In this case, we use Problem 5.19 because it simplifies the reduction proof in Section 5.6.*

5.4 From SBTM to CA1

In this section, we construct a CA1 from an SBTM $M = (Q, \delta)$ and shows that their termination are equivalent. First, we define the alphabet Σ.

Definition 5.23 (Σ_M) *Since M's tape contains only boolean values, it is enough to define the alphabet as*

$$\Sigma_M = {}^\circ Q \times \mathbb{B}.$$

Fact 5.24 Σ_M *is finite and discrete.*

Then we define the local update function f.

Definition 5.25 (f_M) *If* $\delta(q, b) = {}^\circ(q', b', d)$*, then*

$$f_M(x) = \begin{cases} {}^\circ(\emptyset, b') & x = ((\emptyset, b1), ({}^\circ q, b), (\emptyset, b2)) \\ {}^\circ(\emptyset, b) & x = (({}^\circ q, b1), (\emptyset, b), (\emptyset, b2)), d = \texttt{L} \\ {}^\circ({}^\circ q', b) & x = (({}^\circ q, b1), (\emptyset, b), (\emptyset, b2)), d = \texttt{R} \\ {}^\circ({}^\circ q', b) & x = ((\emptyset, b1), (\emptyset, b), ({}^\circ q, b2)), d = \texttt{L} \\ {}^\circ(\emptyset, b) & x = ((\emptyset, b1), (\emptyset, b), ({}^\circ q, b2)), d = \texttt{R} \\ {}^\circ a & x = (a, b, c) \end{cases}$$

otherwise $f_M(x) = \emptyset$.

Fact 5.26 $(\emptyset, \perp)$ *is a quiescent state with respect to* f_M.

Intuitively, if M, on state q and configuration $c = (l, b, r)$, makes a step, then f updates the current CA1 configuration locally as follows:

Case 1 if q is in the center cell, f updates the center cell with the new boolean value,

Case 2 if q is in the left cell and the head of M moves to the left, f leaves the center cell unchanged,

Case 3 if q is in the left cell and the head of M moves to the right, f moves q to the center cell and leaves its boolean value unchanged,

Case 4 if q is in the right cell and the head of M moves to the left, f moves q to the center cell and leaves its boolean value unchanged,

Case 5 if q is in the right cell and the head of M moves to the right, f leaves the center cell unchanged, and

Case 6 if the head of M is not in the neighborhood, then f simply leaves the center cell unchanged.

Formally, we describe the construction in Definition 5.27.

Definition 5.27 (Construction) *Given an SBTM $M = (Q, \delta)$, we define a CA1 $C = (\Sigma, f)$ as follows:*

- *its alphabet Σ is defined in Definition 5.23, which admits a quiescent state $(\emptyset, \bot)$,*
- *its local update function f is defined in Definition 5.25.*

We encode configurations of M into configurations of C as follows.

Definition 5.28 (Encode) *The encoding of an SBTM configuration $c = (l, b, r)$, starting at state q, into a CA1 configuration, written as $[\![c]\!]$, is defined as $([\![l]\!], {}^\circ({}^\circ q, b), [\![r]\!])$ where $[\![\cdot]\!] : \mathcal{L}(\mathbb{B}) \to \mathbb{N} \to \mathcal{O}(\Sigma)$ is defined as follows:*

$$[\![l]\!] = \begin{cases} {}^\circ(\emptyset, \bot) & l = [] \\ {}^\circ(\emptyset, b) & l = b :: l', n = 0 \\ [\![l']\!] & l = b :: l', n > 0 \end{cases}$$

Fact 5.29 *For all l and $n \in \mathbb{N}$, $[\![l]\!](n) \neq \emptyset$.*

Fact 5.30 *For all finite tape t, $[\![t]\!]$ is a finite CA1 configuration.*

Fact 5.31 *For all finite tape t, $[\![t]\!]$ is not a halting CA1 configuration.*

Fact 5.32 *Let $M = (Q, \delta)$. For all tape $t = (l, b, r)$ and $q \in Q$, if $\delta(q, b) = \emptyset$ then $G_f([\![(l, b, r)]\!])$ is a halting CA1 configuration.*

Since SBTM moves its head around during its runs, we need to define a similar operation for a CA1 configuration.

Definition 5.33 (Shift) *We define a function σ that shifts a CA1 configuration $c = (l, m, r)$ one step towards direction $d \in \{\mathsf{left}, \mathsf{right}\}$. In case of $d = \mathsf{left}$, it is defined as follows:*

$$\sigma_{\mathsf{left}}(l, m, r) = (l', m', r')$$

where

$$\begin{aligned} m' &= l(0) \\ l'(n) &= l(n+1) \\ r'(n) &= \begin{cases} m & n = 0 \\ r(n-1) & n > 0 \end{cases} \end{aligned}$$

The case for $d = \mathsf{right}$ is defined as follows:

$$\sigma_{\mathsf{right}}(l, m, r) = (l', m', r')$$

where

$$\begin{aligned} m' &= r(0) \\ r'(n) &= r(n+1) \\ l'(n) &= \begin{cases} m & n = 0 \\ l(n-1) & n > 0 \end{cases} \end{aligned}$$

Definition 5.34 (Inverse direction) *If $d = \textit{left}$ then $\overline{d} = \textit{right}$ and vice versa.*

Fact 5.35 *For all configurations c, $\sigma_{\overline{d}}(\sigma_d(c)) = c$.*

Fact 5.36 *For all configurations c and directions d, if c is halting then $\sigma_d(c)$ is also halting.*

Corollary 5.37 *For all configurations c and directions d, if $\sigma_d(c)$ is halting then c is halting.*

Fact 5.38 (G commutes with σ) *For all directions d, local update function f, and CA1 configurations c, we have that*

$$G_f(\sigma_d(c)) = \sigma_d(G_f(c))$$

Corollary 5.39 *For all $n \in \mathbb{N}$, directions d, local update function f, and CA1 configurations c, we have that*

$$G_f^n(\sigma_d(c)) = \sigma_d(G_f^n(c))$$

Fact 5.40 *Let $M = (Q, \delta)$. For all tape $t = (l, b, r)$ and $q \in Q$, if $\delta(q, b) = {}^\circ(q', b', d)$ then*

$$[\![(q', \theta_d(l, b', r))]\!] = \sigma_d(G_f([\![(q, (l, b, r))]\!]))$$

Finally, we are ready to show that SBTM minimal halting problem reduces to CA1 minimal halting problem.

Lemma 5.41 (Preservation) *For all SBTM M and its input configuration (q, t), we have that if M halts on (q, t) then a CA1 constructed using Definition 5.27 C halts on $[\![(q, t)]\!]$.*

Proof We have that M halts on (q, t) after k steps and we have to show that C halts on $[\![(q, t)]\!]$ after n steps for some n. We proceed by induction on k.

- If $k = 0$, we are done.

- Otherwise, $k = k' + 1$ for some k'. The main idea here is to check whether M halts on $\delta(q, b)$ i.e. after taking a step. If it is, then C halts after one step. Otherwise, we can use the inductive hypothesis to choose the correct number of steps taken by C before it halts.

  In more detail, we have that
  $$S^{k'}(S(q,t))$$
  is a halting TM configuration and we show that
  $$G^n([\![(q,t)]\!])$$
  is a halting configuration, for some n. Let $t = (l, b, r)$ and we proceed by case analysis on $\delta(q, b)$:

  - If $\delta(q, b) = \emptyset$ we chose $n = 1$ and the claim follows from Fact 5.32 and Fact 5.31.
  - Otherwise $\delta(q, b) = {}^\circ(q', b', d)$. By the inductive hypothesis, we have that
    $$G^{k'}([\![(q', \theta(d, (l, b', r)))]\!])$$
    is halting and for all $m < k'$,
    $$G^m([\![(q', \theta(d, (l, b', r)))]\!])$$
    is not halting. We chose $n = k' + 1 = k$. By Fact 5.40 and Definition 5.34 we have
    $$\sigma_{\overline{d}}([\![(q', \theta_d(l, b', r))]\!]) = G_f([\![(q, (l, b, r))]\!])$$
    which we can use to rewrite our assumption (together with Corollary 5.39) to rewrite our proof obligation into showing that
    $$\sigma_{\overline{d}}(G_f^{k'}([\![(q', \theta_d(l, b', r))]\!]))$$
    is a halting configuration. The claim then follows from Corollary 5.37 and the inductive hypothesis.

Lemma 5.42 (Reflection) *For all SBTM M and its input configuration (q, t), we have that if a CA1 constructed using Definition 5.27 C halts on $[\![(q, t)]\!]$ then M halts on (q, t).*

Proof This direction is more straightforward than the previous one, since a minimal halting condition is also a halting condition.

We have that C halts on $[\![(q, t)]\!]$ after n steps and we have to show that M halts on (q, t) after k steps for some k. We chose $k = n$ and we proceed by induction on n.

- If $n = 0$, we have that $[\![(q,t)]\!]$ is already a halting configuration, thus we obtain a contradiction by Fact 5.31.
- Otherwise, $n = n' + 1$ for some n'. We have that

$$G_f^{n'}(G_f([\![(q,t)]\!]))$$

is a halting configuration and we show that

$$S^{n'}(S(q,t))$$

is a halting TM configuration. Let $t = (l,b,r)$ and we proceed by case analysis on $\delta(q,b)$:

  - If $\delta(q,b) = \emptyset$ then we are done.
  - Otherwise $\delta(q,b) = {}^\circ(q',b',d)$. By Fact 5.40 and Definition 5.34 we have

$$\sigma_{\vec{d}}([\![(q',\theta_d(l,b',r))]\!]) = G_f([\![(q,(l,b,r))]\!])$$

which we can use to rewrite our assumption (together with Corollary 5.39) to conclude that

$$\sigma_{\vec{d}}(G_f^{n'}([\![(q',\theta_d(l,b',r))]\!]))$$

is a halting configuration. The claim then follows from Corollary 5.37 and the inductive hypothesis.

Theorem 5.43 *SBTM minimal halting problem reduces to CA1 minimal halting problem.*

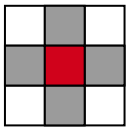
Proof Construction 5.27 is a reduction function by Lemma 5.41 and Lemma 5.42.

5.5 Two-dimensional Cellular Automata

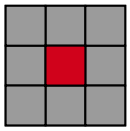
A two-dimensional cellular automata (CA2) is a triple (Σ, N, f) where Σ is a finite and discrete set of states, $N \in (\Sigma^{\mathbb{Z}^2})^n$ is the neighborhood vector, and $f : \Sigma^n \to \Sigma$ is the *local update function* [15, Subsection 2.1]. Given a neighborhood vector $N = (\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n})$, the neighbors of a cell $\vec{x} \in \mathbb{Z}^2$ are the n cells located at $\vec{x} + \vec{x_i}$ for $i = 1, 2, \ldots, n$.

A CA2 configuration is a function $\mathbb{Z}^2 \to \Sigma$ comprising of cells corresponding to each elements of Z^2. A CA2 updates its configurations c using its *global update function* G by simultaneous application of f,

$$G(c)(\vec{x}) = f(c(\vec{x} + \vec{x_1}), c(\vec{x} + \vec{x_2}), \ldots, c(\vec{x} + \vec{x_n})) \tag{5.2}$$



(A) von Neumann neighborhood



(B) Moore neighborhood

FIGURE 5.1: Two common neighborhoods in CA2

for all $i = 1, 2, \ldots, n$.

The two common neighborhoods are the van Neumann and Moore neighborhoods [15, Subsection 2.2], depicted in Figure 5.1, where the neighbors of the red cell are itself *and* the gray cells. For the remainder of this chapter, we assume that N is the von Neumann neighborhood.

Similar CA1 configurations, we formalize CA2 configurations as triples (a, c, b) where $a, b : \mathbb{N} \to \mathbb{N} \to \mathcal{O}(\Sigma)$ and $c : \mathbb{N} \to \mathcal{O}(\Sigma)$. Since for all n, $a(n)$, $b(n)$, and c are also a CA1 configuration, the definition of halting CA2 configuration is straightforward:

Definition 5.44 (CA2 Halting configuration) *A CA2 configuration (a, c, b) is a halting configuration iff either*

- *c is a halting CA1 configuration, or*
- *$\exists n.a(n)$ is a halting CA1 configuration, or*
- *$\exists n.b(n)$ is a halting CA1 configuration.*

Remark 5.45 *As it is the case with CA1, it is undecidable whether a CA2 configuration is halting or not.*

Next, we define a CA2 global transition function G_f with respect to a CA2 local update function f.

Definition 5.46 (next)

$$\mathit{next}((l_1, m_1, r_1), (l, m, r), (l_2, m_2, r_2)) = (l', m', r')$$

where

$$m' = f^{\uparrow}(m_1, l(0), m, r(0), m_2)$$

$$l'(n) = \begin{cases} f^{\uparrow}(l_1(0), l(1), l(0), m, l_2(0)) & n = 0 \\ f^{\uparrow}(l_1(n), l(n+1), l(n), l_2(n-1), l_2(n)) & n > 0 \end{cases}$$

$$r'(n) = \begin{cases} f^{\uparrow}(r_1(0), m, r(0), r(1), r_2(0)) & n = 0 \\ f^{\uparrow}(r_1(n), r(n-1), r(n), r(n+1), r_2(n)) & n > 0 \end{cases}$$

Definition 5.47 (CA1 Global transition function)

$$G_f(a, c, b) = (a', c', b')$$

where

$$c' = \mathit{next}(a(0), c, b(0))$$

$$a'(n) = \begin{cases} \mathit{next}(a(1), a(0), c) & n = 0 \\ \mathit{next}(a(n+1), a(n), a(n-1)) & n > 0 \end{cases}$$

$$b'(n) = \begin{cases} \mathit{next}(c, b(0), b(1)) & n = 0 \\ \mathit{next}(b(n-1), b(n), b(n+1)) & n > 0 \end{cases}$$

Note that G_f is deterministic when f is.

Fact 5.48 *For all CA2 configuration c, if c is a halting configuration then $G_f(c)$ is also a halting configuration.*

Corollary 5.49 *For all $n \in \mathbb{N}$ and CA2 configuration c, if c is a halting configuration then $G_f^n(c)$ is also a halting configuration.*

As with CA1, we modify the definition of quiescent states and finite configurations [16, Subsection 2.2]. Additionally, we introduce a notion of *pseudo-quiescent* states.

Definition 5.50 (Quiescent and pseudo-quiescent states) *A state $q \in \Sigma$ is called a* quiescent *state iff*

$$f(q, q, q, q, q) = {}^{\circ}q.$$

Similarly, given a quiescent state q, state q' is called a pseudo-quiescent *state iff*

$$f(q, q', q', q', q) = {}^{\circ}q'.$$

Remark 5.51 *We introduce the notion of pseudo-quiescent state to simplify the reduction from the halting problem of CA1 to the halting problem of CA2 as we will see in Chapter 5.6. This weakening is justified since a pseudo-quiescent state q' behaves almost the same as a quiescent state, in the sense that q' represents blanks or absence of informations, which means that one can use q' to define a notion of finite configurations (Definition 5.52).*

Definition 5.52 (Finite configuration) *Let q be a quiescent state and q' be a pseudo-quiescent state. We say that (a, c, b) is a spatially-finite CA2 configuration iff all the following criteria hold.*

- *c is a finite CA1 configuration with respect to q'*
- *There exists a bound m such that*

  - *For all $n < m$, $a(n)$ is a finite CA1 configuration with respect to q*
  - *For all $n \geq m$, $a(n)$ is a blank CA1 configuration*

- *There exists a bound m such that*
  - *For all $n < m$, $b(n)$ is a finite CA1 configuration with respect to q*
  - *For all $n \geq m$, $b(n)$ is a blank CA1 configuration.*

Finally, we state the definition of the minimal halting problem of weakly reversible CA2 (Definition 5.53) that we are working with.

Definition 5.53 (Weak reversibility) *Let C be a CA2 and G_f be its associated global transition function. For all CA2 configurations s and t, if $G_f(s) = G_f(t)$ and both $G_f(s)$ and $G_f(t)$ are not halting configurations, then C is reversible.*

Definition 5.54 (CA2 minimal halting) *A CA2 halts on a* finite *configuration c iff there exists $n \in \mathbb{N}$ such that $G_f^n(c)$ is halting and $\forall m < n. G_f^m(c)$ is not halting.*

Problem 5.55 (CA2 minimal halting problem) *Given a CA2 C and an input configuration c, does C halt starting from c?*

5.6 From CA1 to Weakly Reversible CA2

In this section, we construct a weakly reversible (cf. Definition 5.53) CA2 from a CA1. Since CA1 and CA2 share common notations, we distinguish them by prepending a subscripts e.g. ${}_1f$ refer to local update functions of CA1 and ${}_2f$ refer to local update functions of CA2. When it is clear from the context, we drop the pre-subscript to simplify the notation.

Let $C = ({}_1\Sigma, {}_1f)$ be a CA1. Our main idea is, given a CA1 configuration ${}_1c$, to construct a CA2 configuration ${}_2(a, c, b)$ such that $c = {}_1c$, keeping all the rows of ${}_2b$ blank, and using an appropriate CA2 local update function ${}_2f$ to store previous values of ${}_1c$ in the rows in ${}_2a$. Since local update functions can only observe cells in the neighborhood (in particular, they do not now the global structure of CA2 configurations), we need to mark the cells in the CA2 configuration that corresponds with ${}_1c$. This motivates the definition of the CA2 states ${}_2\Sigma$ below.

Definition 5.56 (${}_2\Sigma$)

$$ {}_2\Sigma = \{M, U\} \times {}_1\Sigma $$

For any $q \in {}_1\Sigma$, a CA2 cell $\circ(M, q)$ means that it is marked. Conversely, it is unmarked if its value is $\circ(U, q)$.

Fact 5.57 *If ${}_1\Sigma$ is finite and discrete then ${}_2\Sigma$ is also finite and discrete.*

We define the CA2 local update function ${}_2f$ as follows.

Definition 5.58 (${}_2f$) *Let ${}_1q$ be a quiescent state in ${}_1\Sigma$.*

$$ {}_2f(x) = \begin{cases} [\![{}_1f(q_l, q_m, q_r)]\!] & x = ((U,q_a),(M,q_l),(M,q_m),(M,q_r),(U,q_b)) \\ {}^\circ(U,q) & x = ((M,q_a),(U,q_l),(U,q_m),(U,q_r),(U,q_b)) \\ {}^\circ(U,q_b) & x = ((U,q_a),(U,q_l),(U,q_m),(U,q_r),(x,q_b)) \\ \emptyset & \text{otherwise} \end{cases} $$

where

$$ [\![{}_1f(l,m,r)]\!] = \begin{cases} {}^\circ(M,v) & {}_1f(l,m,r) = {}^\circ v \\ \emptyset & \text{otherwise.} \end{cases} $$

Fact 5.59 *(U,q) is a quiescent state and (M,q) is a pseudo-quiescent state with respect to ${}_2f$.*

Intuitively, for all CA1 configurations ${}_1c$, if ${}_1(G_f)({}_1c)$ is not a halting configuration, then ${}_2f$ ensures that:

Case 1 if the center row is marked (and nothing else is marked) then ${}_2f$ marks the output of ${}_1f$,

Case 2 if the marker is above (and every other cell is blank) then do nothing,

Case 3 if the marker is below or not in the neighborhood, copy the value from below, and

Case 4 otherwise halt.

Formally, we describe the construction in Definition 5.60.

Definition 5.60 (Construction) *Given a CA1 $C = ({}_1\Sigma, {}_1f)$, we define a two-dimensional cellular automata $C2 = ({}_2\Sigma, {}_2f)$ as follows:*

- *its alphabet ${}_2\Sigma$ is defined in Definition 5.56, which given a quiescent state q in ${}_1\Sigma$, admits a quiescent state (U,q) and a pseudo-quiescent state (M,q), and*
- *its local update function ${}_2f$ is defined in Definition 5.58.*

5.6.1 Showing Weak Reversibility

Definition 5.58 ensures that as long as a CA2 configuration c is not a halting configuration, one can recover c back from ${}_2G_f(c)$. We make this fact explicit by defining a local update function ${}_2g$, which acts as an inverse to ${}_2f$, as follows.

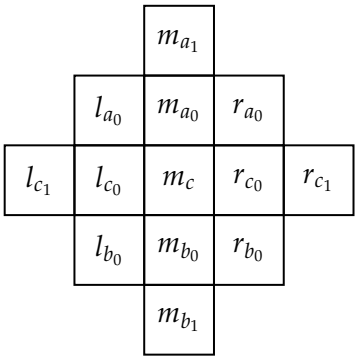


FIGURE 5.2: Fact 5.64, pictorially

Definition 5.61 (${}_2g$)

$$ {}_2g(x) = \begin{cases} {}^\circ(U, q_m) & x = ((M, q_a), (U, q_l), (U, q_m), (U, q_r), (U, q_b)) \\ {}^\circ(M, q_a) & x = ((U, q_a), (M, q_l), (M, q_m), (M, q_r), (U, q_b)) \\ {}^\circ(U, q_a) & x = ((U, q_a), (U, q_l), (U, q_m), (U, q_r), (x, q_b)) \\ \varnothing & \textit{otherwise} \end{cases} $$

It is enough to show Fact 5.64 below to assert that ${}_2G_f$ is invertible, and thus C2 is weakly reversible. Even though Fact 5.64 can be shown using a simple case distinction, it is tedious to do so due to a large number of cases which are ultimately irrelevant due to the assumptions. Instead, using an inductive predicate ${}^\circ f$ (Definition 5.62) below, we characterize the happy paths i.e. those cells where applying ${}_2G_f$ followed by ${}_2G_g$ on them result in a cell containing some $\sigma \in {}_2\Sigma$.

Definition 5.62 (${}^\circ f$) *Let q be a quiescent state with respect to ${}_1f$.*

$$ \frac{}{{}^\circ f({}^\circ(U, a), {}^\circ(M, l), {}^\circ(M, m), {}^\circ(M, r), {}^\circ(U, b), (M, v))} \; f^\circ\text{-CENTER} \quad {}_1f(l, m, r) = {}^\circ v $$

$$ \frac{}{{}^\circ f({}^\circ(U, a), {}^\circ(U, l), {}^\circ(U, m), {}^\circ(U, r), {}^\circ(x, b), (U, b))} \; f^\circ\text{-ABOVE} $$

$$ \frac{}{{}^\circ f({}^\circ(M, a), {}^\circ(U, q), {}^\circ(U, q), {}^\circ(U, q), {}^\circ(U, q), (U, q))} \; f^\circ\text{-BLANK} $$

Intuitively, $f^\circ(i, j, k, l, m, n)$ represents the cases where ${}_2f^\uparrow(i, j, k, l, m) = {}^\circ n$.

Fact 5.63 *For all $a, l, m, r, b, x \in {}_2\Sigma$, if ${}_2f^\uparrow(a, l, m, r, b) = {}^\circ x$ then $f^\circ(a, l, m, r, b, x)$ holds.*

Finally, we can show that ${}_2G_f$ is invertible.

Fact 5.64 (${}_2f$ is invertible) *For all cells l_{c_0}, l_{c_1}, l_{a_0}, l_{b_0}, m_c, m_{a_0}, m_{a_1}, m_{b_0}, m_{b_1}, r_{c_0}, r_{c_1}, r_{a_0}, and r_{b_0} (c.f. Figure 5.2), let*

$$\begin{aligned} m'_{a_0} &= {}_2f^{\uparrow}(m_{a_1}, l_{a_0}, m_{a_0}, r_{a_0}, m_c) \\ l'_{c_0} &= {}_2f^{\uparrow}(l_{a_0}, l_{c_1}, l_{c_0}, m_c, l_{b_0}) \\ m'_c &= {}_2f^{\uparrow}(m_{a_0}, l_{c_0}, m_c, r_{c_0}, m_{b_0}) \\ r'_{c_0} &= {}_2f^{\uparrow}(r_{a_0}, m_c, r_{c_0}, r_{c_1}, r_{b_0}) \\ m'_{b_0} &= {}_2f^{\uparrow}(m_c, l_{b_0}, m_{b_0}, r_{b_0}, m_{b_1}). \end{aligned}$$

If $m'_{a_0} \neq \emptyset$, $l'_{c_0} \neq \emptyset$, $m'_c \neq \emptyset$, $r'_{c_0} \neq \emptyset$, and $m'_{b_0} \neq \emptyset$ then we have

$${}_2g^{\uparrow}(m'_{a_0}, l'_{c_0}, m'_c, r'_{c_0}, m'_{b_0}) = mc.$$

Proof We begin by case distinction on m'_{a_0}, l'_{c_0}, m'_c, r'_{c_0}, and m'_{b_0}, noting that the $\emptyset$ cases directly contradict the assumptions. Otherwise, the claim follows using inversion and Fact 5.63.

Fact 5.65 *For all CA2 configuration c, if $G_f(c)$ is not halting then $G_g(G_f(c)) = c$.*

Proof Follows from Fact 5.64.

Lemma 5.66 *C2 is weakly reversible.*

Proof Let $G_f(s)$ and $G_f(t)$ be non-halting CA2 configurations. By Fact 5.65, we have that $G_g(G_f(s)) = s$ and $G_g(G_f(t)) = t$. Since $G_f(s) = G_f(t)$, it follows that $s = t$.

5.6.2 Showing Simulation

Looking at Definition 5.58, we observe that for each step that C takes during its execution, $C2$ also takes a step as long as its configurations are of certain forms. We make this fact explicit using the a notion of sensible CA2 configurations and a notion of synchronicity between a CA1 configuration and a CA2 configuration.

Definition 5.67 (Sensible configurations) *We say that a row ${}_2(l, m, r)$ is* unmarked *iff*

- $\forall n. \exists {}_1\sigma \in {}_1\Sigma. {}_2l(n) = {}^{\circ}(U, {}_1\sigma)$, *and*
- $\forall n. \exists {}_1\sigma \in {}_1\Sigma. {}_2r(n) = {}^{\circ}(U, {}_1\sigma)$, *and*
- $\exists {}_1\sigma \in {}_1\Sigma. {}_2m = {}^{\circ}(U, {}_1\sigma)$.

Let ${}_1q$ be a quiescent state in ${}_1\Sigma$. A CA2 configuration ${}_2c = {}_2(a, c, b)$ is sensible iff each rows in a and b are unmarked. Additionally, each cells in b contains ${}_1q$.

Definition 5.68 (Sync) *We say that a CA2 configuration ${}_2(a, c, b)$ is in sync with a CA1 configuration ${}_1c$ iff $c = {}_1c$.*

Given an input configuration c to C, we encode it into a CA2 configuration $[\![c]\!]$ and we show that $[\![c]\!]$ is sensible, finite, and in sync with c.

Definition 5.69 (Encode) *Let ${}_1q$ be a quiescent state in ${}_1\Sigma$. The encoding of an CA1 configuration ${}_1c = {}_1(l, m, r)$ into a CA2 configuration, written as $[\![c]\!]$, is defined as*

$$[\![c]\!] = (\lambda\_.\lambda\_.{}^\circ(U, q), (\lambda n.M^\uparrow(l'(n)), M^\uparrow(m'), \lambda n.M^\uparrow(l'(n))), \lambda\_.\lambda\_.{}^\circ(U, q))$$

where

$$M(x) = {}^\circ(M, x)$$

Fact 5.70 *For any CA1 configuration c, $[\![c]\!]$ is sensible and in sync with c.*

Fact 5.71 *For any CA1 configuration c, $[\![c]\!]$ is finite.*

Next, we show that during the course of its run simulating C, $C2$ maintains the invariant that its configurations are sensible and in sync with the corresponding configurations of C.

Fact 5.72 *Let ${}_1c$ be a CA1 configuration and ${}_2c$ be a CA2 configuration. If ${}_1c$ and ${}_2c$ are in sync, ${}_2c$ is sensible, and ${}_1c$ is not a halting configuration, then ${}_1G_f(c)$ and ${}_2G_f(c)$ are in sync. Furthermore, ${}_2G_f(c)$ is also sensible.*

Corollary 5.73 *For all $n \in \mathbb{N}$, CA1 configuration ${}_1c$, and CA2 configuration ${}_2c$, if ${}_1c$ and ${}_2c$ are in sync, ${}_2c$ is sensible, and ${}_1G_f^n(c)$ is not a halting configuration, then ${}_1G_f^n(c)$ and ${}_2G_f^n(c)$ are in sync. Furthermore, ${}_2G_f^n(c)$ is also sensible.*

Remark 5.74 *The assumption that ${}_1c$ is not halting in Fact 5.72 is crucial; without it, we cannot show that ${}_1G_f({}_1c)$ and ${}_2G_f({}_2c)$ are in sync. However, since it is undecidable whether a CA1 configuration is halting or not, we need to get this information from somewhere else. It turns out that the halting problem definitions in Definition 5.19 gives us this information, which simplifies our proof.*

Fact 5.75 *Let ${}_1c$ be a CA1 configuration and ${}_2c$ be a sensible CA2 configuration which is in sync with ${}_1c$. We have that ${}_1c$ is a halting configuration iff ${}_2c$ is a halting configuration.*

Corollary 5.76 *For all $n \in \mathbb{N}$, CA1 configuration ${}_1c$, and CA2 configuration ${}_2c$, if ${}_1c$ and ${}_2c$ are in sync, ${}_2c$ is sensible, and ${}_2G_f^n(c)$ is not a halting configuration, then ${}_1G_f^n(c)$ and ${}_2G_f^n(c)$ are in sync. Furthermore, ${}_2G_f^n(c)$ is also sensible.*

Finally, we are ready to show the reduction from CA1 to weakly-reversible CA2.

Lemma 5.77 (Preservation) *For all CA1 C and CA1 configuration ${}_1c$, if C halts on ${}_1c$ then a CA2 constructed using Definition 5.60 $C2$ halts on $[\![{}_1c]\!]$.*

Proof We have that C halts on ${}_1c$ after n steps such that $\forall m < n$, ${}_1G_f^m(c)$ is not halting and we show that $C2$ halts on $[\![{}_1c]\!]$ after k steps such that $\forall m < k$, ${}_2G_f^m([\![{}_1c]\!])$ is not halting, for some k. We chose $k = n$ and we proceed by case distinction on n. If $n = 0$ then we are done, otherwise $n = n' + 1$ for some n'. By Fact 5.75, it is enough to show that ${}_1G_f^{n'+1}(c)$ and ${}_2G_f^{n'+1}([\![{}_1c]\!])$ are in sync and ${}_2G_f^{n'+1}([\![{}_1c]\!])$ is sensible. This, in turns, follows from Corollary 5.73 and Fact 5.75.

Lemma 5.78 (Reflection) *For all CA1 C and CA1 configuration ${}_1c$, if a CA2 constructed using Definition 5.60 $C2$ halts on $[\![{}_1c]\!]$ then C halts on ${}_1c$.*

Proof We have that $C2$ halts on $[\![{}_1c]\!]$ after n steps such that $\forall m < n$, ${}_2G_f^m([\![{}_1c]\!])$ is not halting and we show that C halts on ${}_1c$ after k steps such that $\forall m < k$, ${}_1G_f^m(c)$ is not halting, for some k. We chose $k = n$ and we proceed by case distinction on n. If $n = 0$ then we are done, otherwise $n = n' + 1$ for some n'. By Fact 5.75, it is enough to show that ${}_1G_f^{n'+1}(c)$ and ${}_2G_f^{n'+1}([\![{}_1c]\!])$ are in sync and ${}_2G_f^{n'+1}([\![{}_1c]\!])$ is sensible. This, in turns, follows from Corollary 5.76 and Fact 5.75.

Theorem 5.79 *The halting problem of CA1 reduces to the halting problem of weakly-reversible CA2.*

Proof Construction 5.60 is a reduction function by Lemma 5.77 and Lemma 5.78.

5.7 Discussion and Related Work

Cellular automata represent a massively parallel computation in contrast to Turing machines or counter machines. Nevertheless, cellular automata have been shown to be computationally universal. Two well-known results in this regard are the universality of Conway's Game of Life [2] and the universality of Wolfram's Rule 110 [27]. Since Rule 110 is a one-dimensional cellular automaton and Toffoli [26] showed that any d-dimensional cellular automaton can be simulated by a $d+1$-dimensional cellular automaton, this implies that 2-dimensional cellular automata are also universal. Morita and Harao [22] later showed a more direct result by constructing a reversible one-dimensional cellular automaton that simulates a reversible Turing machine, which itself is also universal due to Bennett [3]. Regarding the concept of reversibility in cellular automata itself, one can decide whether

a given one-dimensional cellular automaton is reversible or not [1]. However, such results does not extend to two-dimensional cellular automata [14].

To the best of our knowledge, none of the above results have been mechanized before. Furthermore, at the time of writing, we are not aware of any other mechanizations of cellular automata in Coq. As such, we made a number of trade-offs in this work by weakening the notion of reversibility (cf. Definition 5.53) so that we may define termination without trivially circumventing full extensional reversibility. We also use a weaker notion of finite two-dimensional cellular automata configurations by introducing pseudo-quiescent states (Definition 5.50) to simplify the reduction from one-dimensional cellular automata. This weakening is justified because pseudo-quiescent states still enforces (spatial) finiteness while at the same time relaxing the specification of the local update function.

Bibliography

[1] Serafino Amoroso and Yale N. Patt. "Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tessellation Structures". In: *J. Comput. Syst. Sci.* 6.5 (1972), pp. 448–464. DOI: 10.1016/S0022-0000(72)80013-8. URL: https://doi.org/10.1016/S0022-0000(72)80013-8.

[2] AK Austin. "Winning ways (for your mathematical plays), vols 1 and 2, by Elwyn R. Berlekamp, John H. Conway and Richard K. Guy. Pp 469 and 475.£ 10· 80 each. 1982. ISBN 0-12-091101-9/02-7 (Academic Press)". In: *The Mathematical Gazette* 67.441 (1983), pp. 242–243.

[3] Charles H Bennett. "Logical reversibility of computation". In: *IBM journal of Research and Development* 17.6 (1973), pp. 525–532.

[4] Charles H. Bennett. "Notes on Landauer's principle, reversible computation, and Maxwell's Demon". In: *Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics* 34.3 (2003). Quantum Information and Computation, pp. 501–510. ISSN: 1355-2198. DOI: https://doi.org/10.1016/S1355-2198(03)00039-X. URL: https://www.sciencedirect.com/science/article/pii/S135521980300039X.

[5] J. H. Conway. "FRACTRAN: A Simple Universal Programming Language for Arithmetic". In: *Open Problems in Communication and Computation*. Ed. by Thomas M. Cover and B. Gopinath. New York, NY: Springer New York, 1987, pp. 4–26. ISBN: 978-1-4612-4808-8. DOI: 10.1007/978-1-4612-4808-8_2. URL: https://doi.org/10.1007/978-1-4612-4808-8_2.

[6] Martin Davis. "Hilbert's Tenth Problem is Unsolvable". In: *The American Mathematical Monthly* 80.3 (1973), pp. 233–269. URL: https://doi.org/10.1080/0002980.1973.11993265.

[7] Andrej Dudenhefner. "Certified Decision Procedures for Two-Counter Machines". In: *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*. Ed. by Amy P. Felty. Vol. 228. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 16:1–16:18. DOI: 10.4230/LIPIcs.FSCD.2022.16. URL: https://doi.org/10.4230/LIPIcs.FSCD.2022.16.

[8] Yannick Forster. *Computability in constructive type theory*. 2021. DOI: http://dx.doi.org/10.22028/D291-35758.

[9] Yannick Forster, Dominik Kirst, and Gert Smolka. "On Synthetic Undecidability in Coq, with an Application to the Entscheidungsproblem". In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2019. Cascais, Portugal: Association for Computing Machinery, 2019, pp. 38–51. ISBN: 9781450362221. DOI: 10.1145/3293880.3294091. URL: https://doi.org/10.1145/3293880.3294091.

[10] Yannick Forster and Dominique Larchey-Wendling. "Certified undecidability of intuitionistic linear logic via binary stack machines and minsky machines". In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*. Ed. by Assia Mahboubi and Magnus O. Myreen. ACM, 2019, pp. 104–117. DOI: 10.1145/3293880.3294096. URL: https://doi.org/10.1145/3293880.3294096.

[11] Yannick Forster et al. "A Coq Library of Undecidable Problems". In: *The Sixth International Workshop on Coq for Programming Languages*. 2020. URL: https://github.com/uds-psl/coq-library-undecidability.

[12] Michael P. Frank. "Introduction to Reversible Computing: Motivation, Progress, and Challenges". In: *Computing Frontiers. First International Workshop on Reversible Computing*. 2005.

[13] Edward Fredkin and Tommaso Toffoli. "Conservative Logic". In: *Collision-Based Computing*. Ed. by Andrew Adamatzky. Springer, 2002, pp. 47–81. DOI: 10.1007/978-1-4471-0129-1_3. URL: https://doi.org/10.1007/978-1-4471-0129-1%5C_3.

[14] Jarkko Kari. "Reversibility and Surjectivity Problems of Cellular Automata". In: *J. Comput. Syst. Sci.* 48.1 (1994), pp. 149–182. DOI: 10.1016/S0022-0000(05)80025-X. URL: https://doi.org/10.1016/S0022-0000(05)80025-X.

[15] Jarkko Kari. "Theory of cellular automata: A survey". In: *Theor. Comput. Sci.* 334.1-3 (2005), pp. 3–33. DOI: 10.1016/j.tcs.2004.11.021. URL: https://doi.org/10.1016/j.tcs.2004.11.021.

[16] Jarkko Kari and Nicolas Ollinger. "Periodicity and Immortality in Reversible Computing". In: *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*. Ed. by Edward Ochmanski and Jerzy Tyszkiewicz. Vol. 5162. Lecture Notes in Computer Science. Springer, 2008, pp. 419–430. DOI: 10.1007/978-3-540-85238-4_34. URL: https://doi.org/10.1007/978-3-540-85238-4%5C_34.

[17] Rolf Landauer. "Irreversibility and Heat Generation in the Computing Process". In: *IBM J. Res. Dev.* 5.3 (1961), pp. 183–191. DOI: 10.1147/rd.53.0183. URL: https://doi.org/10.1147/rd.53.0183.

[18] Dominique Larchey-Wendling. "Synthetic Undecidability of MSELL via FRACTRAN Mechanised in Coq". In: *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*. Ed. by Naoki Kobayashi. Vol. 195. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 18:1–18:20. DOI: 10.4230/LIPIcs.FSCD.2021.18. URL: https://doi.org/10.4230/LIPIcs.FSCD.2021.18.

[19] Dominique Larchey-Wendling and Yannick Forster. "Hilbert's Tenth Problem in Coq (Extended Version)". In: *Log. Methods Comput. Sci.* 18.1 (2022). DOI: 10.46298/lmcs-18(1:35)2022. URL: https://doi.org/10.46298/lmcs-18(1:35)2022.

[20] Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Automatic computation Bd. 2. Prentice-Hall, 1967. ISBN: 9780131655638. URL: https://books.google.de/books?id=VkRDAAAAIAAJ.

[21] Kenichi Morita. "Universality of a Reversible Two-Counter Machine". In: *Theor. Comput. Sci.* 168.2 (1996), pp. 303–320. DOI: 10.1016/S0304-3975(96)00081-3. URL: https://doi.org/10.1016/S0304-3975(96)00081-3.

[22] Kenichi Morita and Masateru Harao. "Computation universality of one-dimensional reversible (injective) cellular automata". In: *IEICE TRANSACTIONS (1976-1990)* 72.6 (1989), pp. 758–762.

[23] Kenichi Morita and Yoshikazu Yamaguchi. "A Universal Reversible Turing Machine". In: *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*. Ed. by Jérôme Olivier Durand-Lose and Maurice Margenstern. Vol. 4664. Lecture Notes in Computer Science. Springer, 2007, pp. 90–98. DOI: 10.1007/978-3-540-74593-8_8. URL: https://doi.org/10.1007/978-3-540-74593-8%5C_8.

[24] Gert Smolka. *Modeling and Proving in Computational Type Theory Using the Coq Proof Assistant*. https://www.ps.uni-saarland.de/~smolka/drafts/icl2021.pdf (visited 2022-05-11). 2022.

[25] The Coq Development Team. *The Coq Proof Assistant*. Version 8.15. Jan. 2022. DOI: 10.5281/zenodo.5846982. URL: https://doi.org/10.5281/zenodo.5846982.

[26] Tommaso Toffoli. "Computation and Construction Universality of Reversible Cellular Automata". In: *J. Comput. Syst. Sci.* 15.2 (1977), pp. 213–231. DOI: 10.1016/S0022-0000(77)80007-X. URL: https://doi.org/10.1016/S0022-0000(77)80007-X.

[27] Stephen Wolfram. *A New Kind of Science*. Wolfram-Media, 2002. ISBN: 978-1-57955-008-0.