Periodicity and Immortality in Reversible Computing

Jarkko Kari, Nicolas Ollinger

Periodicity and Immortality in Reversible Computing*

Jarkko Kari[1] and Nicolas Ollinger[2]

[1] Department of Mathematics, FIN-20014 University of Turku, Finland,
jkari@utu.fi
[2] Laboratoire d'informatique fondamentale de Marseille (LIF),
Aix-Marseille Université, CNRS,
39 rue Joliot-Curie, 13 013 Marseille, France,
Nicolas.Ollinger@lif.univ-mrs.fr

Abstract. We investigate the decidability of the periodicity and the immortality problems in three models of reversible computation: reversible counter machines, reversible Turing machines and reversible one-dimensional cellular automata. Immortality and periodicity are properties that describe the behavior of the model starting from arbitrary initial configurations: immortality is the property of having at least one non-halting orbit, while periodicity is the property of always eventually returning back to the starting configuration. It turns out that periodicity and immortality problems are both undecidable in all three models. We also show that it is undecidable whether a (not-necessarily reversible) Turing machine with moving tape has a periodic orbit.

Introduction

Reversible computing is the classical counterpart of quantum computing. Reversibility refers to the fact that there is an inverse process to retrace the computation back in time, i.e., the system is time invertible and no information is ever lost. Much of the research on reversible computation is motivated by the Landauer's principal which states a strict lower bound on the amount of energy dissipation which must take place for each bit of information that is erased [1]. Reversible computation can, in principle, avoid this generation of heat.

Reversible Turing machine (RTM) was the earliest proposed reversible computation model [2, 3]. Since then, reversibility has been investigated within other common computation models such as Minsky's counter machines [4, 5] and cellular automata [6]. In particular, reversible cellular automata (RCA) have been extensively studied due to the other physics-like attributes of cellular automata such as locality, parallelism and uniformity in space and time of the update rule.

All three reversible computation models are Turing complete: they admit simulations of universal Turing machines, which naturally leads to various undecidability results for reachability problems. In this work we view the systems,

* Work supported by grants of the French ANR and Academy of Finland # 211967

however, rather differently by investigating their behavior from arbitrary starting configurations. This is more a dynamical systems approach. Each device is understood as a transformation $F : X \longrightarrow X$ acting on its configuration space X. In all cases studied here (counter machines, two Turing machine models – with moving head and with moving tape – and cellular automata) space X is endowed a topology under which F is continuous. In the cases of Turing machines with moving tape and cellular automata, it is the compact and metrizable topology obtained as the enumerable infinite product of the discrete topology on each finite component of a configuration. The action F may be partial, so that it is undefined for some elements of X. Configurations on which F is undefined are called *halting*. We call F *immortal* if there exists a configuration $x \in X$ that never evolves into a halting configuration, that is, $F^n(x)$ is defined for all positive integers n. In contrast, a *mortal* system eventually halts, regardless of the starting configuration. We call F *uniformly mortal* if a uniform time bound n exists such that $F^n(x)$ is not defined for any $x \in X$. If F is continuous, X compact, and the set of halting configurations open then mortality and uniform mortality are equivalent concepts. This means that mortal Turing machines and cellular automata are automatically uniformly mortal. In contrast, a counter machine may be mortal without being uniformly mortal. (A simple example is a one-counter machine where the counter value is repeatedly decremented until it becomes zero and the machine halts.)

Periodicity, on the other hand, is defined for *complete* systems: systems without halting configurations. We call total $F : X \longrightarrow X$ *uniformly periodic* if there is a positive integer n such that F^n is the identity map. *Periodicity* refers to the property that every configuration is periodic, that is, for every $x \in X$ there exists time n such that $F^n(x) = x$. Periodicity and uniform periodicity are equivalent concepts in the cases of cellular automata (Section 3.3) and Turing machines under both modes (Section 2.1), while a counter machine can be periodic without being uniformly periodic (Example 1 in Section 1.1).

In this work we are mainly concerned with decidability of these concepts. Immortality of unrestricted (that is, not necessarily reversible) Turing machines was proved undecidable already in 1966 by Hooper [7]. Our main result (Theorem 7) is a reversible variant of Hooper's approach where infinite searches during counter machine simulations by a Turing machine are replaced by recursive calls to the counter machine simulation itself with empty initial counters. Using reversible counter machines, the recursive calls can be unwound once the search is complete. In a sense this leads to a simpler construction than in Hooper's original article.

Our result also answers an open problem of control theory from [8]. That paper pointed out that if the immortality problem for reversible Turing machines is undecidable, then so is observability for continuous rational piecewise-affine planar homeomorphisms.

As another corollary we obtain the undecidability of the periodicity of Turing machines (Theorem 8). The related problem of determining if a given Turing machine has at least one periodic orbit (under the moving tape mode) is proved

undecidable for reversible, non-complete Turing machines, and for non-reversible, complete Turing machines. The problem remains open under reversible and complete machines. The existence of periodic orbits in Turing machines and counter machines have been investigated before in [9, 10]. Article [9] formulated a conjecture that every complete Turing machine (under the moving tape mode) has at least one periodic orbit, while [10] refuted the conjecture by providing an explicit counter example. The counter example followed the general idea of [7] in that recursive calls were used to prevent unbounded searches. In [10] is was also shown that it is undecidable if a given complete counter machine has a periodic orbit. We show that this is the case even under the additional constraint of reversibility (Theorem 6).

In Theorem 12 we reduce the periodicity problem of reversible Turing machine into the periodicity problem of one-dimensional cellular automata. The immortality problem of reversible cellular automata has been proved undecidable in [11]. Our proofs for the undecidability of immortality (Theorem 1) and periodicity (Theorem 3) among reversible counter machines follow the techniques of [5]. Interestingly, the uniform variants of both immortality and periodicity problems are decidable for counter machines (Theorems 2 and 4).

The paper is organized into three parts dealing with RCM (section 1), with RTM (section 2) and with RCA (section 3). Each part consists of four subsections on (1) definitions, (2) the immortality problem, (3) the periodicity problem, and (4) the existence of periodic orbits. Due to page constraints most proofs are short sketches of the main idea – referees can find full proofs in the appendix.

1 Reversible Counter Machines

1.1 Definitions

Following [5], we define special counter machine instructions for a simpler syntactic characterization of local reversibility and forget about initial and accepting states as we are only interested in dynamical properties.

Let $\Upsilon = \{0, +\}$ be the set of test values and $\Phi = \{-, 0, +\}$ be the set of counter operations whose reverse are defined by $-^{-1} = +$, $0^{-1} = 0$ and $+^{-1} = -$. For all $j \in \mathbb{Z}_k$ and $\phi \in \Phi$, testing τ and modifying $\theta_{j,\phi}$ actions are defined for all $k \in \mathbb{Z}$, $i \in \mathbb{Z}_k$ and $v \in \mathbb{N}^k$ as:

$$\tau(k) = \begin{cases} 0 & \text{if } k = 0 \\ + & \text{if } k > 0 \end{cases} \qquad \theta_{j,\phi}(v)(i) = \begin{cases} v(i) - 1 & \text{if } v(i) > 0,\ i = j \text{ and } \phi = - \\ v(i) & \text{if } i \neq j \text{ or } \phi = 0 \\ v(i) + 1 & \text{if } i = j \text{ and } \phi = + \end{cases}$$

A *k-counter machine* M is a triple (S, k, T) where S is a finite set of states, $k \in \mathbb{N}$ is the number of counters, and $T \subseteq S \times \Upsilon^k \times \mathbb{Z}_k \times \Phi \times S$ is the transition table of the machine. Instruction $(s, u, i, -, t)$ is not allowed in T if $u(i) = 0$. A configuration $\mathfrak{c}$ of the machine is a pair (s, v) where $s \in S$ is a state and $v \in \mathbb{N}^k$ is the value of the counters. The machine can transform a configuration $\mathfrak{c}$ in a configuration $\mathfrak{c}'$ in one step, noted as $\mathfrak{c} \vdash \mathfrak{c}'$, by applying an instruction $\iota \in T$.

An instruction $(s, u, i, \phi, t) \in T$ can be applied to any configuration (s, v) where $\tau(v) = u$ leading to the configuration $(t, \theta_{i,\phi}(v))$. The transitive closure of $\vdash$ is noted as $\vdash^*$.

A counter machine (S, k, T) is a *deterministic k-counter machine (k-DCM)* if at most one instruction can be applied from any configuration. Formally, the transition table must satisfy the following condition:

$$(s, u, i, \phi, t) \in T \wedge (s, u, i', \phi', t') \in T \Rightarrow (i, \phi, t) = (i', \phi', t').$$

The *transition function* of a deterministic counter machine is the function $G : S \times \mathbb{N}^k \to S \times \mathbb{N}^k$ which maps a configuration to the unique transformed configuration, that is for all $(s, v) \in S \times \mathbb{Z}^k$,

$$G(s, v) = \begin{cases} (t, \theta_{i,\phi}(v)) & \text{if } (s, u, i, \phi, t) \in T \text{ and } \tau(v) = u \\ \perp & \text{otherwise} \end{cases}$$

The set of reverse instructions of an instruction is defined as follows:

$(s, u, i, 0, t)^{-1} = \{(t, u, i, 0, s)\}$,
$(s, u, i, +, t)^{-1} = \{(t, u', i, -, s)\}$, where $u'(i) = +, u'(j) = u(j)$ for $j \neq i$,
$(s, u, i, -, t)^{-1} = \{(t, u, i, +, s), (t, u', i, +, s)\}$, where $u'(i) = 0, u'(j) = u(j)$ for $j \neq i$.

The reverse T^{-1} of a transition table T is defined as $T^{-1} = \bigcup_{\iota \in T} \iota^{-1}$. The *reverse* of counter machine $M = (S, T)$ is the machine $M^{-1} = (S, T^{-1})$. A *reversible k-counter machine (k-RCM)* is a deterministic k-counter machine whose reverse is deterministic.

Example 1. The complete DCM $(\{l, l', r, r'\}, 2, T)$ with the following T is locally periodic but not periodic: $T = \{ (l, (0, *), 0, 0, r), (r, (*, 0), 1, 0, l), (l, (+, *), 0, -, l'), (r, (*, +), 1, -, r'), (l', (*, *), 1, +, l), (r', (*, *), 0, +, r) \}$ ($*$: any value). In states l, l' tokens are moved from the first counter to the second, and in states r, r' back to the first counter. Its reverse is obtained by swapping $l \leftrightarrow r$ and $l' \leftrightarrow r'$. □

1.2 Undecidability of the Immortality Problem

Theorem 1. *It is undecidable whether a given 2-RCM is immortal.*

Proof sketch. By [7] the immortality problem is undecidable among 2-CM, while [5] provides an effective immortality/mortality preserving conversion of an arbitrary k-CM into a 2-RCM. ■

Remark. The 2-RCM constructed in the proof through Morita's construction [5] can be forced to have mortal reverse. This is obtained by adding in the original CM an extra counter that is being continuously incremented.

Theorem 2. *It is decidable whether a given k-CM is uniformly mortal.*

Proof sketch. Induction on k: The claim is trivial for $k = 0$. For the inductive step, let M be a k-CM, $k \geq 1$. For $i = 1, 2, \ldots, k$ set counter i to be always positive and test whether the so obtained $(k-1)$-CM M_i is uniformly mortal. If all k recursive calls return a positive answer, set n to be a common uniform mortality time bound for all k machines M_i. Since counters can be decremented by one at most, we know that configurations of M with some counter value $\geq n$ are mortal. Immortality hence occurs only if there is a period within the finite number of configurations with all counters $< n$. ■

1.3 Undecidability of the Periodicity Problem

Theorem 3. *It is undecidable whether a given 2-RCM is periodic.*

Proof sketch. Let $M = (S, 2, T)$ be a given 2-RCM whose reverse is mortal. In particular, there are no periodic configurations in M. According to the remark after Theorem 1 it is enough to effectively construct a complete 2-RCM M' that is periodic if and only if M is mortal. Machine M' has state set $S \times \{+, -\}$ where states $(s, +)$ and $(s, -)$ represent M in state s running forwards or backwards in time, respectively. In a halting configuration the direction is switched. ■

Analogously to Theorem 2 one can prove the following result.

Theorem 4. *It is decidable whether a given k-CM is uniformly periodic.*

1.4 Periodic Orbits

Theorem 5 ([10]). *It is undecidable whether a given complete 2-DCM admits a periodic configuration.*

Theorem 6. *It is undecidable whether a given complete 3-RCM admits a periodic configuration, and it is undecidable whether a given (not necessarily complete) 2-RCM admits a periodic configuration.*

Proof sketch. We first prove the result for complete 3-RCM. The construction in [5] shows that it is undecidable for a given 2-RCM $M = (S, 2, T)$ without periodic configurations and two given states s_1 and s_2 whether there are counter values n_1, n_2, m_1 and m_2 such that $(s_1, n_1, m_1) \vdash^* (s_2, n_2, m_2)$. By removing all transitions from state s_2 and all transitions into state s_1 we can assume without loss of generality that all configurations (s_1, n_1, m_1) and (s_2, n_2, m_2) are halting in M^{-1} and M, respectively. Using a similar idea as in the proof of Theorem 3 we effectively construct a 3-RCM $M' = (S \times \{+, -\}, 3, T')$ that simulates M forwards and backwards in time using states $(s, +)$ and $(s, -)$, respectively, and counters 1 and 2. The direction is switched at halting configurations. In addition, counter 3 is incremented at halting configurations, except when the state is s_1 or s_2.

Machine M' is clearly reversible and complete. Moreover, since M has no periodic configurations, the only periodic configurations of M' are those where

M is simulated back and forth between states s_1 and s_2. This completes the proof for 3-RCM.

Using the construction of [5] a three counter RCM can be converted into a 2-RCM and that conversion preserves periodic orbits. ■

The 2-RCM provided by the construction in [5] is not complete. It seems likely that it can be modified to give a complete 2-RCM, but details remain to be worked out:

Conjecture 1. It is undecidable whether a given complete 2-RCM admits a periodic configuration.

2 Reversible Turing Machines

2.1 Definitions

The classical model of Turing machines consider machines with a moving head (a configuration is a triple $(s, z, c) \in S \times \mathbb{Z} \times \Sigma^{\mathbb{Z}}$). Following Kůrka [9], we consider machines with a moving tape as our base model to endow the space of configurations with a compact topology. Following [5], we define two kinds of instructions for a simpler syntactic characterization of local reversibility.

Let $\Delta = \{\leftarrow, \rightarrow\}$ be the set of directions with inverses $(\leftarrow)^{-1} = \rightarrow$ and $(\rightarrow)^{-1} = \leftarrow$. For all $\delta \in \Delta$ and $a \in \Sigma$, moving σ_δ and writing μ_a actions are defined for all $c \in \Sigma^{\mathbb{Z}}$ and $z \in \mathbb{Z}$ as:

$$\sigma_\delta(c)(z) = \begin{cases} c(z+1) \text{ if } \delta = \rightarrow \\ c(z-1) \text{ if } \delta = \leftarrow \end{cases} \qquad \mu_a(c)(z) = \begin{cases} a & \text{if } z = 0 \\ c(z) & \text{if } z \neq 0 \end{cases}$$

A *Turing machine* M is a triple (S, Σ, T) where S is a finite set of states, Σ is a finite set of symbols, and $T \subseteq S \times \Delta \times S \cup S \times \Sigma \times S \times \Sigma$ is the transition table of the machine. A configuration $\mathfrak{c}$ of the machine is a pair (s, c) where $s \in S$ is a state and $c \in \Sigma^{\mathbb{Z}}$ is the content of the tape. The machine can transform a configuration $\mathfrak{c}$ in a configuration $\mathfrak{c}'$ in one step, noted as $\mathfrak{c} \vdash \mathfrak{c}'$, by applying an instruction $\iota \in T$. An instruction $(s, \delta, t) \in T \cap S \times \Delta \times S$ is a *move instruction* of the machine, it can be applied to any configuration (s, c), leading to the configuration $(t, \sigma_\delta(c))$. An instruction $(s, a, t, b) \in T \cap S \times \Sigma \times S \times \Sigma$ is a *matching instruction* of the machine, it can be applied to any configuration (s, c) where $c(0) = a$, leading to the configuration $(t, \mu_b(c))$.

A Turing machine (S, Σ, T) is a *deterministic Turing machine (DTM)* if at most one instruction can be applied from any configuration. Formally, the transition table must satisfy the following conditions:

$$\begin{aligned} (s, \delta, t) \in T \wedge (s', a', t', b') \in T &\Rightarrow s \neq s' \\ (s, \delta, t) \in T \wedge (s, \delta', t') \in T &\Rightarrow \delta = \delta' \wedge t = t' \\ (s, a, t, b) \in T \wedge (s, a, t', b') \in T &\Rightarrow t = t' \wedge b = b' \end{aligned}$$

The *local transition function* of a DTM is the function $f : S \times \Sigma \to S \times \Delta \cup S \times \Sigma \cup \{\perp\}$ defined for all $(s, a) \in S \times \Sigma$ as follows. The associated partial *global*

transition function $G : S \times \Sigma^{\mathbb{Z}} \to S \times \Sigma^{\mathbb{Z}}$ maps a configuration to the unique transformed configuration, that is for all $(s, c) \in S \times \Sigma^{\mathbb{Z}}$,

$$f(s,a) = \begin{cases} (t,\delta) \text{ if } (s,\delta,t) \in T \\ (t,b) \text{ if } (s,a,t,b) \in T \\ \bot \quad \text{otherwise} \end{cases} \qquad G(s,c) = \begin{cases} (t, \sigma_\delta(c)) \text{ if } f(s,c(0)) = (t,\delta) \\ (t, \mu_b(c)) \text{ if } f(s,c(0)) = (t,b) \end{cases}$$

Lemma 1. *If all configurations of a DTM are periodic or mortal then there is a uniform bound n such that for all configurations (s, c) either $G^n(s, c)$ is undefined or $G^t(s, c) = (s, c)$ for some $0 < t < n$. In particular, a periodic DTM is uniformly periodic and a mortal DTM is uniformly mortal.*

Proof. For every $n > 0$ let $U_n = \{(s, c) \mid G^n(s, c) = (s, c)$ or $G^n(s, c)$ undef$\}$ be the set of configurations that are mortal or periodic at time n. Sets U_n are open so $U_1, U_2, \ldots$ is an open cover of the compact set of all configurations. It has a finite subcover. ■

One might think that periodicity characterizes a different set of machines if one considers Turing machines with a moving head instead of a moving tape but it is not the case. The global transition function with moving head $H : S \times \mathbb{Z} \times \Sigma^{\mathbb{Z}} \to S \times \mathbb{Z} \times \Sigma^{\mathbb{Z}}$ is defined so that for each $(s, z, c) \in S \times \mathbb{Z} \times \Sigma^{\mathbb{Z}}$, $H(s, z, c) = (s', z', c')$ where $G(s, \sigma^z_{\rightarrow}(c)) = (s', \sigma^{z'}_{\rightarrow}(c'))$. A DTM is periodic with moving head if for each configuration $\mathfrak{c}$, there exists $t \in \mathbb{N}$ such that $H^t(\mathfrak{c}) = \mathfrak{c}$ or equivalently if there exists some $t \in \mathbb{N}$ such that $H^t = \mathrm{Id}$.

Lemma 2. *A DTM is periodic if and only if it is periodic with moving head.*

Proof. Assume that Σ has at least two elements. For each $t \in \mathbb{N}$ and $(s, z, c) \in S \times \mathbb{Z} \times \Sigma^{\mathbb{Z}}$, $H^t(s, z, c) = (s', z', c')$ where $G^t(s, \sigma^z_{\rightarrow}(c)) = (s', \sigma^{z'}_{\rightarrow}(c'))$. Thus, if $H^t = \mathrm{Id}$ then $G^t = \mathrm{Id}$. Conversely, let $G^t = \mathrm{Id}$. By definition, $H^t(s, z, c) = (s, z', c')$ for some z' such that $\sigma^z_{\rightarrow}(c) = \sigma^{z'}_{\rightarrow}(c')$. Moreover, as the machine acts locally, for all d and k such that $c_{|[z-t,z+t]} = d_{|[k-t,k+t]}$, $H^t(s, k, d) = (s, k + z' - z, d')$ where $d' = \sigma^{z'-z}_{\rightarrow}(d')$. If $z' - z \neq 0$, one might choose d such that $d(k + t(z' - z)) \neq d(k + (t + 1)(z' - z))$, contradicting the hypothesis. Thus, $H^t = \mathrm{Id}$. ■

The reverse of an instruction is defined as follows: $(s, \delta, t)^{-1} = (t, \delta^{-1}, s)$ and $(s, a, t, b)^{-1} = (t, b, s, a)$. The reverse T^{-1} of a transition table T is defined as $T^{-1} = \left\{\iota^{-1} \middle| \iota \in T\right\}$. The *reverse* of Turing machine $M = (S, \Sigma, T)$ is the machine $M^{-1} = (S, \Sigma, T^{-1})$. A *reversible Turing machine (RTM)* is a deterministic Turing machine whose reverse is deterministic.

Lemma 3. *It is decidable whether a given Turing machine is reversible.*

Proof. It is sufficient to syntactically check the transition table. ■

Lemma 4. *The reverse of a mortal RTM is mortal.*

Proof. The uniform bound is valid for both the mortal RTM and its reverse. ■

Lemma 5. *The reverse of a complete RTM is a complete RTM. In particular, a complete RTM is surjective.*

Proof. A DTM is complete if and only if $n|\Sigma| + m = |S||\Sigma|$ where n and m are the numbers of move and matching instructions, respectively. The claim follows from the fact that M and M^{-1} always have the same numbers of move and matching instructions. ■

2.2 Undecidability of the Immortality Problem

Theorem 7. *It is undecidable whether a given RTM is immortal.*

Proof sketch. For a given 2-RCM without periodic configurations, and given initial state s_0, we effectively construct a reversible Turing machine that is mortal if and only if the 2-RCM halts from the initial configuration $(s_0, 0, 0)$. The Theorem then follows from [5], where it was shown that the halting problem is undecidable for 2-RCM. Note that our additional constraint that the 2-RCM has no periodic configurations can be easily established by having an extra counter that is incremented on each step of the counter machine. This counter can then be incorporated in the existing two counters with the methods of [5].

As a first step we do a fairly standard simulation of a 2-CM by a TM. Configuration (s, a, b) where s is a state and $a, b \in \mathbb{N}$ is represented as a block "`@1`^{a}`x2`^{b}`y`" of length $a + b + 3$, and the Turing machine is positioned on the symbol "`@`" in state s. A simulation of one move of the CM consists of (1) finding delimiters "`x`" and "`y`" on the right to check if either of the two counters is zero, and (2) incrementing or decrementing the counters as determined by the CM. The TM is then returned to the beginning of the block in the new state of the CM. If the CM halts then also the TM halts. All this can be done reversibly if the simulated CM is reversible.

The TM constructed as outline above has the problem that it has immortal configurations even if the CM halts. These are due to the unbounded searches for delimiter symbols "`@`", "`x`" or "`y`". Searches are needed when testing whether the second counter is zero, as well as whenever either counter is incremented or decremented.

Unbounded searches lead to infinite searches if the symbol is not present in the configuration. (For example, searching to the right for symbol "`x`" when the tape contains "`@111`...".) To prevent such infinite searches we follow the idea of [7], also employed in [10]. Instead of a straightforward search using a loop, the search is done by performing a recursive call to the counter machine from its initial configuration $(s_0, 0, 0)$. More precisely, we first make a bounded search of length three to see if the delimiter is found within next three symbols. If the delimiter is not found, we start a recursive simulation of the CM by writing "`@xy`" over the next three symbols, step on the new delimiter symbol "`@`", and enter the initial state s_0. This begins a nested simulation of the CM.

In order to be able to continue the higher level execution after returning from the recursive search, the present state of the TM needs to be written on the tape

when starting the recursive call. For this purpose we increase the tape alphabet by introducing several variants "$\texttt{@}_\alpha$" of the start delimiter "@". Here α is the Turing machine state at the time the search was begun. When returning from a successful recursive search, the higher level computation can pick up from where it left off by reading the state α from the delimiter "$\texttt{@}_\alpha$".

If the recursive search procedure finds the delimiter this is signalled by reversing the search. Once returned to the beginning, the three symbol initial segment "@xy" is moved three positions to the right and the process is repeated. The repeated applications of recursive searches, always starting the next search three positions further right, will eventually bring the machine on the delimiter it was looking for, and the search is completed.

On the other hand, if the CM halts during a recursive search then the TM halts. This always happens when a sufficiently long search is performed using a CM that halts from its initial configuration.

With some additional tricks one can make the TM outlined above reversible, provided the CM is reversible. Now we reason as follows: If the initial configuration $(s_0, 0, 0)$ is immortal in the CM then the TM has a non-halting simulation of the CM. So the TM is not mortal. Conversely, suppose that the CM halts in k steps but the TM has an immortal configuration. The only way for the TM not to halt is to properly simulate the CM from some configuration (s, a, b), where the possibilities $a = \infty$ and $b = \infty$ have to be taken into account. Since the CM has no periodic configurations, one of the two counters necessarily obtains arbitrarily large values during the computation. But this leads to arbitrarily long recursive searches, which is not possible since each such search halts within k steps. ■

Remarks. (1) The RTM constructed in the proof has no periodic configurations. So the undecidability of the immortality problem holds among RTM without any periodic configurations. (2) Add to the 2-RCM a new looping state s_1 in which the first counter is incremented indefinitely. We can also assume without loss of generality that the 2-RCM halts only in state s_2. Then the RTM constructed in the proof has computation $(s_1, c_1) \vdash^* (s_2, c_2)$ for some $c_1, c_2 \in \Sigma^{\mathbb{Z}}$ if and only if the 2-RCM halts from the initial configuration $(s_0, 0, 0)$.

These detailed observations about the proof will be used later in the proofs of Theorems 8 and 9.

2.3 Undecidability of the Periodicity Problem

Theorem 8. *It is undecidable whether a given complete RTM is periodic.*

Proof sketch. For a given RTM $A = (S, \Sigma, T)$ without periodic configurations we effectively construct a complete RTM $A' = (S \times \{+, -\}, \Sigma, T')$ that is periodic if and only if every configuration of A is mortal. States $(s, +)$ and $(s, -)$ of A' are used to represent A in state s running forwards or backwards in time, respectively. In a halting configuration the direction is switched. The result now follows from Theorem 7 and the first remark after its proof. ■

2.4 Periodic Orbits

Theorem 9. *It is undecidable whether a given (non-complete) RTM admits a periodic configuration.*

Proof. Remark (2) after the proof of Theorem 7 pointed out that it is undecidable for a given RTM $A = (S, \Sigma, T)$ without periodic configurations, and two given states $s_1, s_2 \in S$ whether there are configurations (s_1, c_1) and (s_2, c_2) such that $(s_1, c_1) \vdash^* (s_2, c_2)$. By removing all transitions from state s_2 and all transitions into state s_1 we can assume without loss of generality that all configurations (s_1, c_1) and (s_2, c_2) are halting in A^{-1} and A, respectively. Using a similar idea as in the proof of Theorem 8 we effectively construct an RTM $A' = (S \times \{+, -\}, \Sigma, T')$ in which A is simulated forwards and backwards in time using states $(s, +)$ and $(s, -)$, respectively. But now the direction is swapped from "-" to "+" only in state s_1, and from "+" to "-" in state s_2. In other halting situations of A, also A' halts. Clearly $((s_1, +), c_1)$ is periodic in A' if and only if $(s_1, c_1) \vdash^* (s_2, c_2)$ for some $c_2 \in \Sigma^{\mathbb{Z}}$. No other periodic orbits exist in A'. ■

Theorem 10. *It is undecidable whether a given complete DTM admits a periodic configuration.*

Proof. In [10] a complete DTM over the binary tape alphabet was provided that does not have any periodic configurations. This easily gives an analogous DTM for any bigger tape alphabet. For a given RTM $A = (S, \Sigma, T)$ we effectively construct a complete DTM that has a periodic configuration if and only if A has a periodic configuration. The result then follows from Theorem 9. Let $B = (S', \Sigma, T')$ be the fixed complete DTM without periodic configurations from [10], $S \cap S' = \emptyset$. The complete DTM we construct has state set $S \cup S'$ and its transitions includes $T \cup T'$, and in addition a transition into a state $s' \in S'$ whenever A halts. It is clear that the only periodic configurations are those that are periodic already in A. ■

Conjecture 2. A complete RTM without a periodic point exists. Moreover, it is undecidable whether a given complete RTM admits a periodic configuration.

3 Reversible Cellular Automata

3.1 Definitions

A *one-dimensional cellular automaton* A is a triple (S, r, f) where S is a finite *state set*, $r \in \mathbb{N}$ is the *neighborhood radius* and $f : S^{2r+1} \longrightarrow S$ is the *local update rule* of A. Elements of $\mathbb{Z}$ are called cells, and a configuration of A is an element of $S^{\mathbb{Z}}$ that assigns a state to each cell. Configuration c is turned into configuration c' in one time step by a simultaneous application of the local update rule f in the radius r neighborhood of each cell:

$$c'(i) = f(c(i-r), c(i-r+1), \ldots, c(i+r-1), c(i+r)) \text{ for all } i \in \mathbb{Z}.$$

Transformation $G : c \mapsto c'$ is the *global transition function* of A. The Curtis-Hedlund-Lyndom -theorem states that a function $S^{\mathbb{Z}} \longrightarrow S^{\mathbb{Z}}$ is a global transition function of some CA if and only if it is continuous and commutes with the shift σ, defined by $\sigma(c)_i = c_{i+1}$ for all $c \in S^{\mathbb{Z}}$ and $i \in \mathbb{Z}$.

Cellular automaton A is called *reversible* if the global function G is bijective and its inverse G^{-1} is a CA function. We call A *injective*, *surjective* and *bijective* if G is injective, surjective and bijective, respectively. Injectivity implies surjectivity, and bijectivity implies reversibility. See [6] for more details on these classical results.

3.2 Undecidability of the Immortality Problem

Let some states of a CA be identified as halting. Let us call a configuration c *halting* if $c(i)$ is a halting state for some i. We call c *locally halting* if $c(0)$ is a halting state. These two definitions reflect two different ways that one may use to define an accepting computation in CA: either acceptance happens when a halting state appears somewhere, in an unspecified cell, or one waits until a halting state shows up in a fixed, predetermined cell. A configuration c is immortal (locally immortal) for G if $G^n(c)$ is not halting (locally halting, respectively) for any $n \geq 0$. CA function G is immortal (locally immortal) if there exists an immortal (locally immortal) configuration.

Theorem 11 ([11]). *It is undecidable whether a given reversible one-dimensional CA is immortal (locally immortal).*

3.3 Undecidability of the Periodicity Problem

In cellular automata periodicity and uniform periodicity are equivalent. Indeed, suppose that a period n that is common to all configurations does not exist. Then for every $n \geq 1$ there is $c_n \in S^{\mathbb{Z}}$ such that $G^n(c_n) \neq c_n$. Each c_n has a finite segment p_n of length $2rn+1$ that is mapped in n steps into a state that is different from the state in the center of p_n. Configuration c that contains a copy of p_n for all n, satisfies $G^n(c) \neq c$ for all n, and hence such c is not periodic.

Theorem 12. *It is undecidable whether a given one-dimensional CA is periodic.*

Proof sketch. For a given complete reversible Turing machine $M = (S, \Sigma, T)$ we effectively construct a one-dimensional reversible CA $A = (Q, 2, f)$ that is periodic if and only if M is periodic. The result then follows from Theorem 8. The state set

$$Q = \Sigma \times ((S \times \{+, -\}) \cup \{\leftarrow, \rightarrow\})$$

consists of two tracks: The first track stores elements of the tape alphabet Σ and it is used to simulate the content of the tape of the Turing machine, while the second track stores the current state of the simulated machine at its present location, and arrows $\leftarrow$ and $\rightarrow$ in other positions pointing towards the position of the Turing machine on the tape. The arrows are needed to prevent several

Turing machine heads accessing the same tape location and interfering with each other's computation. The state is associated a symbol '+' or '-' indicating whether the reversible Turing machine is being simulated forwards or backwards in time. The direction is switched if the Turing machine sees a local error, i.e., an arrow pointing away from the machine.

It follows from the reversibility of M that A is a reversible CA. If M has a non-periodic configuration c then A has a non-periodic configuration which simulates the computation from c. Conversely, if M is periodic it is uniformly periodic under the moving head mode. It easily follows that all configurations of A are periodic. ■

A one-dimensional RCA is equicontinuous if and only if it is periodic, so we have

Corollary 1. *It is undecidable whether a given one-dimensional reversible CA is equicontinuous.*

3.4 Periodic Orbits

Every cellular automaton has periodic orbits so the existence of periodic orbits is trivial among cellular automata.

References

1. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development **5** (1961) 183–191
2. Lecerf, Y.: Machines de turing réversibles. C. R. Acad. Sci. Paris **257** (1963) 2597–2600
3. Bennett, C.B.: Logical reversibility of computation. IBM Journal of Research and Development **17**(6) (1973) 525–532
4. Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall, Englewoods Cliffs (1967)
5. Morita, K.: Universality of a reversible two-counter machine. Theor. Comput. Sci. **168**(2) (1996) 303–320
6. Kari, J.: Theory of cellular automata: a survey. Theor. Comput. Sci. **334** (2005) 3–33
7. Hooper, P.K.: The undecidability of the turing machine immortality problem. J. Symb. Log. **31**(2) (1966) 219–234
8. Collins, P., van Schuppen. J. H.: Observability of hybrid systems and turing machines. In: 43rd IEEE Conference on Decision and Control. Volume 1., IEEE Press (2004) 7 – 12
9. Kůrka, P.: On topological dynamics of turing machines. Theor. Comput. Sci. **174**(1-2) (1997) 203–216
10. Blondel, V.D., Cassaigne, J., Nichitiu, C.M.: On the presence of periodic configurations in turing machines and in counter machines. Theor. Comput. Sci. **289**(1) (2002) 573–590
11. Kari, J., Lukkarila, V.: Some undecidable dynamical properties for one-dimensional reversible cellular automata. To appear (2008)

A Full Proofs

These appendices contains material for the referees. Appendix A contains full proofs for some sketched proofs omitted due to page constraints. Appendix B contains basic syntax and reference to further material developed by the authors for programming reversible turing machines and simulate the machine of theorem 7.

Proof of Theorem 1 (RCM immortality)

Hooper's article [7] contains a comment that the immortality problem of Turing machines can be reduced to the immortality problem of counter machines. But there is a simple direct argument for the undecidability of counter machine immortality, based on an idea from [10]: For a given 2-CM $M = (S, 2, T)$ and initial state $s_0 \in S$ one effectively construct the following 4-CM M' that is mortal if and only if M halts from the initial configuration $(s_0, 0, 0)$.

Machine M' performs bounded simulations of M using 4 counters: Counters 1 and 2 represent the two counters of M. Counters 3 and 4 bound the length of the simulation of M: Tokens are moved from counter 3 to counter 4 and back. Simulation of M is restarted from the initial configuration $(s_0, 0, 0)$ whenever counter 3 becomes empty. The length of the simulation cycle is $n_3 + n_4$, i.e. the total number of tokens in counters 3 and 4. Whenever the simulation is restarted a new token is added to counter 4, so that the length of the simulation cycle keeps increasing. It is clear that M' has an immortal configuration if and only if configuration $(s_0, 0, 0)$ is immortal in M.

Due to Minsky's classical result on counter machines [4] the halting problem of 2-CM from initial configuration $(s_0, 0, 0)$ is known to be undecidable — hence we conclude that it is undecidable whether a given 4-CM is immortal.

Using the techniques of [5] the counter machine can effectively be made reversible and the number of counters can effectively be reduced to two without changing the mortality status of the machine. Hence Theorem 1 follows.

■

Proof of Theorem 4 (decidability of uniform periodicity in CM)

Let M be a given k-CM. We prove using induction on k that an algorithm exists that tests M for uniform periodicity and that returns a period.

1° If $k = 0$ then the configuration space if finite and can be effectively checked.

2° (Inductive step) Let $k > 0$ and suppose that an algorithm exists to check the unform periodicity of given $(k-1)$-CM. For every $i = 1, 2, \ldots k$, construct the $(k-1)$-CM M_i obtained from M by removing counter i and pretending in all transitions that counter i has always positive value. This corresponds to the idea of having a very large value in counter i. Clearly, if M is uniformly periodic then M_i are uniformly periodic. So we recursively check if all M_i are uniformly periodic. If any one is not, we can conclude that M is not uniformly periodic either.

Suppose then that each M_i is uniformly periodic with period p_i. Note that this does not yet imply that all configurations of M with large value in counter i are periodic – it can namely happen that the value in counter i gets incremented or decremented in otherwise periodic orbits. To remove that possibility we set counter i to value p_i, set other $k-1$ counters to values $\leq p_i$ (and repeat the test for all combinations of such values in all counters) and test whether after p_i iterations of M counter i has returned to value p_i. If not, we know that M is not uniformly periodic.

Next we set $p = \mathrm{lcm}(p_1, p_2, \ldots, p_k)$ so that we know that all configurations where some counter has value $\geq p$ is periodic with period p. All that remains to be done is to check for every configuration c with all counters $< p$ that either c is periodic with period p or that c is periodic in such a way that all counter values remain $< p$ through the iteration starting at c. ■

Proof of Theorem 7 (RTM immortality)
In the core of the paper, we gave a sketch of the proof involving the proper transformation of a given 2-RCM into a RTM. We will explain here the details of the construction and how it should be precisely done so that the obtained RTM verifies all hypothesis.

We provide the reader with two main arguments. First, we describe the syntactic form of the part of the tape already visited by the head of the machine since the beginning of the computation. Second, we describe precisely the machine by giving building blocks and their invariants.

Terra cognita The machine is constructed in such a way that the *terra cognita*, the part of the configuration that has been visited by the machine head since the beginning of the computation, always (re)enters a k-depth well-parenthesis word S as described by the following grammar in a bounded number of steps.

$$
\begin{aligned}
S &\leftarrow S_1 \mid S_2 \mid S_{\underline{1}2} \mid S_{1\underline{2}} \\
S_1 &\leftarrow \mathtt{1}^* P_k \mathtt{1}^* \mid \mathtt{@}_\alpha \mathtt{1}^* P_k \mathtt{1}^* \\
S_2 &\leftarrow \mathtt{2}^* P_k \mathtt{2}^* \mid \mathtt{2}^* P_k \mathtt{2}^* \mathtt{y} \\
S_{\underline{1}2} &\leftarrow \mathtt{1}^* P_k \mathtt{1}^* \mathtt{x} \mathtt{2}^* \mid \mathtt{@}_\alpha \mathtt{1}^* P_k \mathtt{1}^* \mathtt{x} \mathtt{2}^* \mid \mathtt{1}^* P_k \mathtt{1}^* \mathtt{x} \mathtt{2}^* \mathtt{y} \\
S_{1\underline{2}} &\leftarrow \mathtt{1}^* \mathtt{x} \mathtt{2}^* P_k \mathtt{2}^* \mid \mathtt{@}_\alpha \mathtt{1}^* \mathtt{x} \mathtt{2}^* P_k \mathtt{2}^* \mid \mathtt{1}^* \mathtt{x} \mathtt{2}^* P_k \mathtt{2}^* \mathtt{y} \\
P_k &\leftarrow \mathtt{@}_\alpha \mathtt{1}^* P_{k-1} \mathtt{1}^* \mathtt{x} \mathtt{2}^* \mathtt{y} \mid \mathtt{@}_\alpha \mathtt{1}^* \mathtt{x} \mathtt{2}^* P_{k-1} \mathtt{2}^* \mathtt{y} \\
P_0 &\leftarrow \mathtt{@}_\alpha \underline{\mathtt{x}} \mathtt{y}
\end{aligned}
$$

Moreover, the only local transformations the head does on *terra cognita* are:

- replace `111` or `222` by $\mathtt{@}_\alpha\mathtt{xy}$ *(add a new level)*;
- replace `1x2` by `11x` *(during increment counter 1)*;
- replace `2y1` or `2y2` by `22y` *(increment counter)*;
- replace $\mathtt{@}_\alpha\mathtt{xy}$ by `111` or `222` as expected by surroundings *(remove a level)*;
- replace `11x` by `1x2` as expected by surroundings *(during decrement counter 1)*;
- replace `22y` by `2y1` or `2y2` as expected by surroundings *(decrement counter)*;

These transformations ensure *terra cognita* is always well formed.

Constructing the machine We start with some hypothesis on checking and from there build counter machine simulator and effective checking machine. All the constructed machines are reversible and halt on unspecified configurations. The syntax for the following descriptions use the Gni programming language described in appendix B. The program and simulators can be found following instructions in appendix B.

Checking 1 When it returns, $[s|\text{check}_1|t\rangle$ verifies $s.\ \underline{\mathtt{@}_\alpha}\mathtt{1}^m\mathtt{x} \vdash \underline{\mathtt{@}_\alpha}\mathtt{1}^m\mathtt{x}, t$. If it does not return, either it diverges on $s.\ \underline{\mathtt{@}_\alpha}\mathtt{1}^\omega$ or it halts.

Checking 2 When it returns, $[s|\text{check}_2|t\rangle$ verifies $s.\ \underline{\mathtt{x}}2^n\mathtt{y} \vdash \underline{\mathtt{x}}2^n\mathtt{y}, t$. If it does not return, either it diverges on $s.\ \underline{\mathtt{x}}2^\omega$ or it halts.

Bounded search Bounded search is the key part of the construction calling checking recursively.

Bounded search 1 When it does not halt, $[s|\text{search}_1|t_0, t_1, t_2\rangle$ verifies $s.\ \underline{\mathtt{@}_\alpha}1^m\mathtt{x} \vdash \mathtt{@}_\alpha 1^m\underline{\mathtt{x}}, t$ where $k = m \mod 3$ or diverges into check_1 on $s.\ \underline{\mathtt{@}_\alpha}1^\omega$. Notice that if the first call to check_1 in the loop returns, all subsequent calls will also return and the search will eventually return.

```
1  def [s|search_1|t_0, t_1, t_2⟩ :
2     s. @_α ⊢ @_α, l
3     l. →, u
4     u. x ⊢ x, t_0
5      | 1x ⊢ 1x, t_1
6      | 11x ⊢ 11x, t_2
7      | 111 ⊢ 111, c
8     call [c|check_1|p⟩ from 1
9     p. 111 ⊢ 111, l
```

Bounded search 2 When it does not halt, $[s|\text{search}_2|t_0, t_1, t_2\rangle$ verifies $s.\ \underline{\mathtt{x}}2^n\mathtt{y} \vdash \mathtt{x}2^n\underline{\mathtt{y}}, t$ where $k = n \mod 3$ or diverges into check_2 on $s.\ \underline{\mathtt{x}}2^\omega$. Notice that if the first call to check_2 in the loop returns, all subsequent calls will also return and the search will eventually return.

```
1  def [s|search_2|t_0, t_1, t_2⟩ :
2     s. x ⊢ x, l
3     l. →, u
4     u. y ⊢ y, t_0
5      | 2y ⊢ 2y, t_1
6      | 22y ⊢ 22y, t_2
7      | 222 ⊢ 222, c
8     call [c|check_2|p⟩ from 2
9     p. 222 ⊢ 222, l
```

CM instruction The counter machine simulation is quite standard, one has to ensure that bounded searches are used and that the instruction machines are reversible.

CM instruction test counter 1 $[s|\text{test1}|z, p\rangle$ verifies $s.\ \underline{\mathtt{@}_\alpha}\mathtt{x} \vdash \underline{\mathtt{@}_\alpha}\mathtt{x}, z$ and $s.\ \underline{\mathtt{@}_\alpha}1 \vdash \underline{\mathtt{@}_\alpha}1, p$.

```
1  def [s|test1|z, p⟩ :
2     s. @_α x ⊢ @_α x, z
3      | @_α 1 ⊢ @_α 1, p
```

CM instruction test counter 2 $[s|\text{test2}|z, p\rangle$ verifies $s.\ \underline{@_\alpha}1^n\mathtt{xy} \vdash \underline{@_\alpha}1^n\mathtt{xy}, z$ and $s.\ \underline{@_\alpha}1^n\mathtt{x2} \vdash \underline{@_\alpha}1^n\mathtt{x2}, p$ and diverges into check_1 on $s.\ \underline{@_\alpha}1^\omega$.

```
1  def [s|endtest2|z, p⟩ :
2     s. x y ⊢ x y, z
3      | x 2 ⊢ x 2, p
```

```
1  def [s|test2|z, p⟩ :
2     [s|search_1|t_0, t_1, t_2⟩
3     [t_0|endtest2|z_0, p_0⟩
4     [t_1|endtest2|z_1, p_1⟩
5     [t_2|endtest2|z_2, p_2⟩
6     ⟨z_0, z_1, z_2|search_1|z]
7     ⟨p_0, p_1, p_2|search_1|p]
```

CM instruction increment/decrement counter 1/2 The principle is the same as form testing: we use bounded search and ensure reversibility. The full details are given in the GNIRUT toolkit.

CM simulator After initializing the tape, just use the instruction machines using one small trick on test instructions to ensure reversibility if the CM is reversible.

Checking Checking simply consists in simulating CM until collision and then reverting back to initial position.

```
1  fun [s|check_α|t⟩ :
2     [s|RCM_α|h_1, h_2, ...⟩
3     ⟨h_1, h_2, ...|RCM_α|t]
```

When it does not halt, $[s|\text{check}_1|t\rangle$ verifies $s.\ \underline{@_\alpha}1^m\mathtt{x} \vdash \underline{@_\alpha}1^m\mathtt{x}, t$ or diverges on (or halts on a prefix of) $s.\ \underline{@_\alpha}1^\omega$. If it diverges, it executes $[s|\text{RCM}_\alpha|h_1, h_2, \ldots\rangle$ on segments of unbounded size, thus the RCM does not halt.

Symmetrically, when it does not halt, $[s|\text{check}_2|t\rangle$ verifies $s.\ \underline{\mathtt{x}}2^n\mathtt{y} \vdash \underline{\mathtt{x}}2^n\mathtt{y}, t$ or diverges on (or halts on a prefix of) $s.\ \underline{\mathtt{x}}2^\omega$. If it diverges, it executes $[s|\text{RCM}_\alpha|h_1, h_2, \ldots\rangle$ on segments of unbounded size, thus the RCM does not halt.

Proof of Theorem 8 (RTM periodicity)

For a given RTM $A = (S, \Sigma, T)$ we effectively construct a complete RTM $A' = (S \times \{+, -\}, \Sigma, T')$ that is periodic if and only if every configuration of A is either periodic or mortal. States $(s, +)$ and $(s, -)$ are used to represent A in state s running forwards or backwards in time, respectively. In a halting configuration the direction is switched. More precisely, let f and f^{-1} be the local transition functions of A and A^{-1}, respectively. Then the transition table T' is constructed so that the local transition function f' of A' is

$$f'((s,+),a) = \begin{cases} ((t,+),\delta) & \text{if } f(s,a) = (t,\delta) \\ ((t,+),b) & \text{if } f(s,a) = (t,b) \\ ((s,-),a) & \text{if } f(s,a) = \bot \end{cases}$$

$$f'((s,-),a) = \begin{cases} ((t,-),\delta) & \text{if } f^{-1}(s,a) = (t,\delta) \\ ((t,-),b) & \text{if } f^{-1}(s,a) = (t,b) \\ ((s,+),a) & \text{if } f^{-1}(s,a) = \bot \end{cases}$$

It is easy to see that A' is complete and reversible. If all configurations are periodic or mortal in A then they are also periodic or mortal in A^{-1}. It follows that all configurations are periodic in A'. Conversely, suppose that (s, c) is a configuration of A that is neither periodic nor mortal. Then $((s, +), c)$ is not periodic in A'. We conclude that A' is periodic if and only if all configurations of A are either periodic or mortal. According to the first remark after Theorem 7 we may assume that A has no periodic configurations. In this case periodicity of A' is equivalent to mortality of A, and the result follows from Theorem 7. ■

Proof of Theorem 12 (RCA periodicity)

For a given complete reversible Turing machine $M = (S, \Sigma, T)$ we effectively construct a one-dimensional reversible CA $A = (Q, 2, f)$ that is periodic if and only if M is periodic. The state set

$$Q = \Sigma \times ((S \times \{+, -\}) \cup \{\leftarrow, \rightarrow\})$$

consists of two tracks: The first track stores elements of the tape alphabet Σ and it is used to simulate the content of the tape of the Turing machine, while the second track stores the current state of the simulated machine at its present location, and arrows $\leftarrow$ and $\rightarrow$ in other positions pointing towards the position of the Turing machine on the tape. The arrows are needed to prevent several Turing machine heads accessing the same tape location and interfering with each other's computation. The state is associated a symbol '+' or '-' indicating whether the reversible Turing machine is being simulated forwards or backwards in time.

The local update rule f only can change the state of a cell whose radius-one neighborhood contains a Turing machine state on the second track. Let i be a cell that contains Turing machine state on the second track, and let $i + \delta$ be the position of the Turing machine after its next move, where $\delta \in \{-1, 0, 1\}$. Note that the next move is forward or backward in time depending on whether the symbol associated with the state is '+' or '-', respectively.

The move consists of possibly changing the tape symbol on the first track in position i, writing the new state of M in cell $i + \delta$ and writing a left or right arrow in position i depending on whether $\delta = -1$ or $\delta = +1$, respectively. These instructions are as indicated by the local transition function of the Turing machine.

But the move is executed in the CA only if the tape looks locally correct, that is, the nearby cells have arrows pointing towards the TM. More precisely, the second track of cells $i - 1$ and $i + 1$ must contain a right and left arrow, respectively, and

- if $\delta = -1$ then position $i - 2$ contains a right arrow, and
- if $\delta = +1$ then position $i + 2$ contains a left arrow.

These conditions guarantee that the surrounding arrows correctly point to the Turing machine head before and after the move. If any of these conditions is not satisfied then instead of the regular move by the Turing machine, the symbol '+' or '-' is swapped so that the direction of the simulation is reversed.

It follows from the reversibility of M that A is a reversible CA. If M is not periodic then it has a non-periodic configuration $(s, c) \in S \times \Sigma^{\mathbb{Z}}$. Clearly A then has a non-periodic configuration whose first tract reads c and the second track reads $^{\omega} \rightarrow s \leftarrow^{\omega}$.

Conversely, assume that M is periodic, with period p. Let $c \in Q^{\mathbb{Z}}$ be an arbitrary configuration of A. Configuration c is only changed around cells that contain a TM state, but there may be any number of such cells. However, due to

the left and right arrows of the second track, different Turing machine simulations cannot interfere with each other: All activity is constrained within segments of the form $\rightarrow^n s \leftarrow^m$ where $n, m \in \mathbb{N} \cup \{\infty\}$ and $s \in S$. In each such segment the Turing machine either acts periodically with period p, or before time p reaches the end of the segment and reverses. As the inverse TM is also periodic with period p, it is clear that in the second case another reverse will take place and the action is periodic with period at most $2p$. Since all segments are periodic with period at most $2p$, the CA is periodic with period $(2p)!$.

The result now follows from Theorem 8. ■

B Programming with RTM

In order to precisely define and test our constructions for theorem 7, the need for a specific programming language quickly came to our mind. In order to achieve these goal, we developed the GNIRUT toolkit. The toolkit and precise informations on how to use it, plus sample programs associated to theorem 7 can be found at the following address:

`http://www.lif.univ-mrs.fr/~nollinge/rec/gnirut/`

To facilitate reading of the construction for theorem 7, we copy here the main syntax description from the GNIRUT manual:

B.1 Basics statements

A GNI program is a sequence of statements. Each statement is written on a separate line. As we will see when discussing macros, the indentation matters in a GNI program. Comments can be added to statements: starting by a `#` symbol, the comment run until the end of the line. Comments are just ignored by the interpreter.

In a GNI program, states and symbols are represented by identifiers: non-empty sequences of letters (lowercase or uppercase, case-sensitive), digits, underscore, dot and prime (more precisely, any sequence matching the regular expression `[.a-zA-Z0-9_']*[a-zA-Z0-9_']`). The set of states and symbols of the machine defined by a program is the set of states and symbols appearing in its statements.

The syntax of basic statements is the following:

`s. <-, s'` move instruction $(s, \leftarrow, s')$;
`s. ->, s'` move instruction $(s, \rightarrow, s')$;
`s. a:b, s'` matching instruction (s, a, b, s').

When not describing precise syntax, we will give programs in a stylized form like in the following example. Notice that the line numbers are just given to help reading and should not be typed.

1 *begin.* `x:x`, *search*
2 *search.* $\rightarrow$, *loop*
3 *loop.* _:_, *search*
4 *loop.* `x:x`, *end*

B.2 Compound statements

To simplify the writing of matching statements, several matching with the same initial state can be merged into a compound statement. With the same philosophy, default behavior can be defined to facilitate the writing of compound

matchings of the kind *replace a by b and enter state t but if the letter is not a neither b or c then replace it by x and enter state u*. The syntax of compound statements is the following:

`s. a:b, t | b:c, u | d:e, v` sequence of matching instructions (s, a, b, t), (s, b, c, u), (s, d, e, v);

`s. a:b, s' else t` same as before but with a default behavior: on any letter not defined in the compound matching instructions, do not modify the letter but jump to state t: it is syntactically equivalent to replace `else t` by $|w : w, t$ for each letter w of the program (already defined or appearing in latter statements) not already appearing in the statement;

`s. a:b, s' else t but b,c` same as an `else` modifier but the modifier does not apply for letters appearing after the `but` keyword;

`s. a:b, s' else t write a` same as an `else` modifier but instead of not modifying the letter, any letter is replace by a: syntactically like $|w : a, t$;

`s. a:b, s' else t write a but b,c` is a combination of all the modifiers.

B.3 Using macros

As such, the language is clearly sufficient to encode any machine but it can be boring to repeat over and over the same patterns, copies of the same sub-machine performing a given task like *go to the first x on the right*. To help on this, the language provides a macro definition system.

Defining a macro is really the same as defining the main machine. The same instructions are used. In both cases what you obtain a Turing machine. But for macro definitions all states loose their names after definition but the one you select as relevant to the outside world. Moreover, the machine associated to the macro is not added to the current main machine but stored for future use. Once the macro defined, to use it, you ask the interpreter to add a fresh copy of the machine states in the current environment.

Indentation plays a big role in macro definition. After the `def` starting the macro definition, all the lines corresponding to that macro should BE indented by the same amount of spaces and/or tabulations, which should be more than the indentation of the `def` line. When the indentation goes back to the original level, the macro definition ends.

The syntax for macro definition and usage is the following:

`def [i1,...,im|toto|o1,...,on>:` begin the definition of a macro called `toto`; the only states that will be visible from outside are i_1, ..., i_m and o_1, ..., o_n. There is no technical distinction between i and o but it is good practice to consider i as input states and o as output states, to facilitate reading of source code.

`[a,b|titi|c,d,e>` inserts a copy of the macro machine `titi` in the current machine definition: the fresh copy of the machine `titi` will use a, b as input states for its i_1, i_2 and c, d, e as output states for its o_1, o_2, o_3.

The following code defines a macro and uses it:

```
1  def [s|search|t⟩ :
2      s. x:x, u
3      u. →, r
4      r. _:_, u | x:x, t
5
6  [s|search|a⟩
7  [a|search|b⟩
8  b. →, c
9  c. a:b, d | b:a, d
```

B.4 Applying transforms

Sometimes, one would like to reuse a machine with slight syntactical transformations. Currently, two such transforms are defined in Gni. The syntax and usage is the following:

`[a,b|lr titi|c,d,e>` inserts a copy of the macro machine `titi` where the move instruction have inverted directions: ← is replaced with → and reciprocally.

`<o1,...,on|toto|i1,...,im]` inserts a copy of the inverse of the machine `toto`, provided that `toto` is reversible.

B.5 Hooper's style recursive calls

Macros are fine but co-recursive machines would lead to infinite state machine, which is forbidden. One way to avoid this is to use the tape of the Turing machine as a place to push entry point information before calling a machine and popping from that same place to know where to go back. Gni provides that kind of feature, with the following assumptions. It is the programmers responsibility to ensure that the called machine will never modify the tape cell from which it starts and will come back to that particular position at the end of the computation. The syntax and usage is the following:

`fun [i1,...,im|toto|o1,...,on>:` defines a callable function: the syntax and principle are the same as for defining macros but the obtained machine is not usable as a macro, it is used through calls and just one copy of it will be put into the main machine if it is called from at least one place.

`call [i1,...,im|toto|o1,...,on> from a` insert a function call: when entering states i_1 to i_m, the machine will push entry point information on the tape, replacing the letter a that is written on it; on return from the call it will pop the entry point information, replace it by a and change state to $o_1,\ldots,o_n$ depending on the return state. The function does not need to exist at the time the call is inserted, so co-recursive calls are allowed.

`call [i1,...,im|lr toto|o1,...,on> from a` calls the `lr` version of `toto`.

`call <o1,...,on|toto|i1,...,im] from a` calls the inverse version of `toto`.
`call <o1,...,on|lr toto|i1,...,im] from a` calls the inverse of the `lr` version of `toto`.
`@link` issuing a `call` instruction in the interpreter does not effectively modify the machine but stores calling informations. The `@link` command is taking care of wiring all calls, putting copies of used functions and `lr` and/or inverse of them as appropriate, recursively inserting all needed functions that might be called by inserted functions.

Table 1: The following code defines a function and uses it to recursively compute a recursive function on positive integers represented as blocks of x

```
1  def [s|test|z, p⟩ :
2      s. →, r
3      r. _:_, za | x:x, pa
4      za. ←, z
5      pa. ←, p
6
7  def [s|dec|t⟩ :
8      s. →, a
9      a. _:_, b | x:x, s
10     b. ←, c
11     c. x:_, r
12     r. ←, u
13     u. _:_, t | x:x, r
14
15 def [s|copy|t⟩ :
16     s. →, a
17     a. _:_, b | x:o, s
18     b. ←, c
19     c. o:o, b else d
20     d. →, e
21     e. _:_, t | o:x, f
22     f. →, g
23     g. _:_, h | o:o, f
24     h. →, i
25     i. _:x, j | x:x, h
26     j. ←, k
27     k. _:_, b | x:x, j
28
29 fun [s|f|t⟩ :
30     [s|test|t, p⟩
31     [p|copy|a⟩
32     [a|dec|b⟩
33     call [b|f|c⟩ from _
34     c. _:x, l
35     l. ←, o
36     o. x:x, l else t
37
38 @link [s|f|t⟩
```