On Immortal Configurations in Turing Machines

Emmanuel Jeandel

On Immortal Configurations in Turing Machines

Emmanuel Jeandel

LIRMM, CNRS UMR 5506 - CC 477
161 rue Ada, 34 095 Montpellier Cedex 5, France
jeandel@lirmm.fr

Abstract We investigate the immortality problem for Turing machines and prove that there exists a Turing Machine that is immortal but halts on every recursive configuration. The result is obtained by combining a new proof of Hooper's theorem [11] with recent results on effective symbolic dynamics.
Keywords: Turing machines, Dynamical Systems, Π_1^0 Classes, Symbolic Dynamics, Immortality.

In this paper we investigate the behaviour of Turing machines seen as dynamical systems. In this context, a Turing machine does not start from a specified initial state and tape, but from any configuration. In this context, we call a configuration *immortal* if the machine runs forever starting from it. The seminal article of Hooper [8] proves that we cannot decide whether a Turing machine has an immortal configuration (is immortal). However, the result does not say anything about what the immortal configurations look like. In fact, in the construction by Hooper, if the Turing machine has an immortal configuration, then it has one where the tape is almost completely empty.

A few results give some structure on the set of immortal configurations: Kurka [13] asked whether there always exists a (temporally) *periodic* configuration, and was refuted by Blondel et al. [1]. Delvenne and Blondel [6] proved that the set of immortal configurations, if nonempty, always contains quasi-periodic configurations.

In this article, we go further, and give some news results on the set of immortal configurations. We prove in particular in Theorem 2 that there exists a Turing machine for which the set of immortal configurations is nonempty and contains no computable configurations.

The main ingredient is a new proof of Hooper's theorem by Kari and Ollinger [11] combined with an encoding into subshifts of an effective set with no recursive points. We first define all relevant vocabulary in the next section, then proceed to the proof.

1 The Immortality Problem

1.1 Definitions

We first give some definitions and properties of Turing machines. Unless specified, our Turing machines will have only one biinfinite tape. We use the *moving-tape* convention: the head is always at position 0 and the tape, rather than the head, is moving. In this context, the state of the system can be described only by the state of the Turing machine and the content of the tape.

Let Σ be the alphabet of the tape, and S the set of states of the Turing machine M. A *configuration* (also called instantaneous description) is an element of $\mathcal{C} = S \times \Sigma^{\mathbb{Z}}$. The Turing machine can now be seen as a partial function (as an halting configuration has no image) on $\mathcal{C}$.

A configuration $c \in \mathcal{C}$ is *immortal* for M if M runs forever starting from the configuration c. We denote by $I(M)$ the immortal configurations (if they exist) of M. The Turing machine is therefore a dynamical system on $I(M)$.

It is important to note that a configuration is not only a tape, but the combination of a tape and a state. As a consequence, one cannot restrict itself to the dynamics of the Turing machine starting from some specific state, but must take into account computations starting from *any* state.

The first known result is given by Hooper:

Theorem 1 ([8]). *There is no algorithm that decides, given a Turing Machine M, whether $I(M)$ is empty.*

It's important to note that the proof of Hooper and subsequent proofs [11] do not give any insights about the structure of $I(M)$. Indeed, in these proofs, $I(M)$, if not empty, always contains a *finite* configuration (ie every cell of the tape, except a finite number, contains the same symbol). We will prove here that $I(M)$ can be more intricate, and in particular might contain only nonrecursive configurations.

1.2 Effective sets

To understand the computational properties of $I(M)$, we need the following notion:

Definition 1. *A subset $S \subseteq \{0,1\}^{\mathbb{N}}$ is an* effective set *(also called a Π_1^0-set) if there exists an oracle Turing machine M so that S is exactly the set of oracles on which the Turing machine, starting from the initial state and the empty tape, runs forever.*

There exist many equivalent definitions. For example, S is effective if there exists a recursive set of finite words L so that S is exactly the set of infinite words containing no prefix in L. There exists an extensive litterature on effective sets, and we refer the reader to [3,4].

Now, up to a slight (recursive) encoding of the set of configurations $\mathcal{C}$, $I(M)$ can be seen as a subset of $\{0,1\}^{\mathbb{N}}$, and it is quite clear it is effective: it is easy to build a Turing machine that, using $c \in \mathcal{C}$ as on oracle, simulates M on input c.

From the fact that $I(M)$ is effective, we already obtain many properties about its structure, see e.g. the basis theorems in [4]. We also known [10] that there exist (nonempty) effective sets with no recursive points (A point $w \in \{0,1\}^{\mathbb{N}}$ is recursive if there is a Turing machine that outputs w_n given n).

So now, the question is as follows: Are the immortal sets as rich as the effective sets ? To answer this question, we will look at the Turing degree of points of an immortal set. If x and y are two infinite words (or configurations), we say that $x \leq_T y$ if there exists a Turing machine that outputs x given as an oracle. The Turing degree of a word is then its equivalence class for the relation $\leq_T \cap \geq_T$. The degree of recursive points is usually denoted $\mathbf{0}$.

Our first observation is that there is no way to encode any effective sets into immortal sets, preserving e.g. the Turing degrees, due to the following lemma:

Lemma 1 ([13]). *If $I(M)$ is nonempty, then one of the following is true:*

- *it contains a configuration $c \in I(M)$ so that M, starting from c, never reads the symbols in any position $i < 0$ of the tape of c.*
- *it contains a configuration $c \in I(M)$ so that M, starting from c, never reads the symbols in any position $i > 0$ of the tape of c.*

A reformulation is that a Turing machine with moving tape is never positively expansive, see [13]. As a consequence, take the configuration $c \in I(M)$ given by the lemma for which the Turing Machine, wlog, never reads any cell in any position $i < 0$ of c. Then all configurations $c' \in \mathcal{C}$ identical with c on position $i \geq 0$ are also immortal configurations. Choosing the other bits of c' wisely proves we can find in $I(M)$ configurations of any Turing degree greater than the degree of (the right part of) c.

Proposition 1. *If $I(M)$ is nonempty, there exists a Turing degree d so that $I(M)$ contains configurations of any Turing degree above d.*

As there exist nonempty effective sets where any two different points have incomparable Turing degrees[10], there exist effective sets that cannot be encoded as immortal sets. So we need a weaker definition. The good notion is Muchnik equivalence[17]:

Definition 2. *Two subsets S_1 and S_2 of $\{0,1\}^{\mathbb{N}}$ are* Muchnik equivalent *if for every $x_1 \in S_1$, there is a point $x_2 \in S_2$ computable with oracle x_1, and conversely.*

Intuitively, two sets S_1 and S_2 are Muchnik equivalent if they contain the same "minimal" Turing degrees. In particular, S_1 contains a recursive point if and only if S_2 contains a recursive point. See [17] for more information on mass problems and Muchnik equivalence.

We now can state our result:

Theorem 2 (main result). *For every effective set S, there exists a Turing machine M so that $I(M)$, the set of immortal configurations of M, is Muchnik equivalent to S.*

Corollary 1 *There exists a Turing machine M so that $I(M)$ is nonempty and contains only nonrecursive configurations.*

1.3 Effective subshifts

Before going to the proof, we need another ingredient, coming from symbolic dynamics. To introduce this notion, we look at *traces* of Turing machines: For a configuration c, the *trace* of c is the word $u \in (\Sigma \times S)^{\mathbb{N}}$ where u_i contains the letter in position 0 of the tape and the state at the i-th step during the execution of M on input c. The trace is well defined only on configurations $c \in I(M)$ (otherwise u would be finite). Let $T(M)$ be the set of traces of M.

Now we look at the map from c to its trace $u(c)$. First of all, it is clear that $u(c)$ is computable from c. Furthermore, we can reconstruct the cells of the tape of c read by the Turing machine from $u(c)$ (which, depending of c, might or not be all the cells). In particular, there exists a configuration d so that $u(d) = u(c)$ that is computable in $u(c)$: Let $I = [a, b] \subseteq \mathbb{N}$ (possibly with $a = -\infty$ or $b = +\infty$) be the set of cells of c that are visited during the computation of M, and reconstruct the tape of d on I using the trace $u(c)$ and take all other cells to be b for some arbitrary symbol $b \in \Sigma$. In particular, we just have proven that $I(M)$ and $T(M)$ are Muchnik-equivalent. (It is important to note that the interval I is not always computable given $u(c)$: therefore the transformation from $u(c)$ to d is not uniform in $u(c)$. (In technical terms, we have proven that $I(M)$ and $T(M)$ are Muchnik equivalent, not that they are Medvedev equivalent [17]).)

$T(M)$ has interesting properties. It is an effective set: by a compactness argument, $u \in T(M)$ if for every length $n > 0$, there exists a configuration $c \in \mathcal{C}$ so that u coincides with the (possibly finite) trace of c on positions $i < n$. $T(M)$ is also closed under shift: If $u \in T(M)$ then $\sigma(u)$ defined by $\sigma(u)_i = u_{i+1}$ is also in $T(M)$. This means $T(M)$ is what is called an *effective subshift*:

Definition 3. *A subset $S \subseteq \Sigma^{\mathbb{N}}$ is an effective (right-sided) subshift if it is effective and closed under shift.*

An equivalent definition is as follows:

Definition 4. *Let $L \subseteq \Sigma^\star$. We denote by $S(L), S^+(L), S^-(L)$ respectively the set of biinfinite , right infinite, left infinite words over Σ that contain no factor in L.*

Then $S \subseteq \Sigma^{\mathbb{Z}}$ (resp. $\Sigma^{\mathbb{N}}, \Sigma^{\mathbb{Z}^-}$) is an effective twosided (resp. right-sided, left-sided) subshift if $S = S(L)$ (resp. $S^+(L), S^-(L)$) for some recursive language L.

Effective subshifts are an important tool for understanding computability in dynamical systems, and has received increasing attention in recent years [14,18,2]. In particular we obtained a result similar to Proposition 1 for subshifts in [9].

Based on these properties, it is reasonable to try to encode an effective subshift, rather than an effective set, as an immortal set. To be able to do this, we need the following:

Theorem 3 ([14]). *For every effective set S, there exists a language $L \subseteq \{0,1\}^\star$ so that $S(L)$ (resp. $S^+(L), S^-(L)$) is Muchnik-equivalent to S.*

(Proposition 3.1 in [14] is given only for $S(L)$ but it is not hard to see it works for left-sided and two-sided subshifts as well).

2 Proof of the main result

Now we are able to explain how the proof will work. We will start from an effective subshift $S(L)$ given by a recursive set L, and code it into a Turing machine. The machine will have two tracks: The second one will be read-only and contain a biinfinite word w, and the machine will try to prove that $w \in S(L)$. However due to Lemma 1, the Turing machine might on some inputs prove only that some infinite prefix (resp. suffix) of w is in $S^-(L)$ (resp. $S^+(L)$).

To do this, we will start from the proof of the undecidability of the immortality problem by Kari and Ollinger [11], and explain how to modify it for our purposes. In particular, a thorough examination of [11] by the reader is encouraged. We will try as much as possible to use the same notations.

2.1 Counter machines

Usual proofs of the undecidability of the immortality problem usually start with a counter machine. There will be no difference here. However note that we need these machines to accept tapes, i.e. infinite words. For this to make sense, we will consider oracle counter machines. To simplify the definition, we will suppose that the oracle is over the binary alphabet $\{0, 1\}$.

In an oracle counter machine, one of the counter (the first here, for reasons soon to be apparent) is used to represent the position inside the oracle. Informally, an oracle counter machines thus contain three types of instructions: lookup instructions (test if a counter is nonzero), oracle lookup (test if the letter of the oracle is nonzero) and modifying instructions (increase/decrease a counter).

We now define it formally. Let $\Phi = \{-1, 0, 1\}$ and $\Upsilon = \{0, 1\}$.

An oracle k-counter machine is given by a tuple (S, k, T) where S is a finite set of states, $k \in \mathbb{N}$ is the number of counters, and $T : S \times \{1, \dots, k, o\} \times \Upsilon \times \Phi \times S$ the transition relation.

Let w be an infinite word over $\{0, 1\}$ that will be used as oracle for the machine. A configuration of a k-counter machine is an element of $S \times \mathbb{N}^k$. An instruction (s, i, u, v, s') can be applied only on configurations of the form (s, c) and will lead to (s', c') (denoted by $(s, c) \vdash (s', c')$) with respect to the following rules:

- If $i \neq o$, the instruction can only be applied if $u = \min(1, c_i)$. That is, u is the result of the test whether the i-th counter is empty. If $i = o$, the instruction can only be applied if $w_{c_1} = u$, that is if the letter in position c_1 of w is u.
- The instruction will then update the counter c_i depending on v. If $i = o$, it does nothing ($c' = c$). Otherwise $c'_j = c_j$ for $j \neq i$ and $c'_i = c_i + v$. Note that it is incorrect to decrease the value of a counter that is already zero.

Note that the first counter plays a special role. A counter machine computes by applying instructions. If at some point there is no instruction that can be applied, then the machine halts. We will only be interested in deterministic counter machines (DCM), that is for which the map $(s, i, u, v, s') \mapsto (s, u)$ is injective.

Let s_0 be a special (initial) state of the counter machine. We say that a word $w \in \{0,1\}^{\mathbb{N}}$ is accepted by a counter machine if, starting from $(s_0, 0^k)$, the computation of the counter machine never halts. If we follow the usual encoding of Turing machines into 3-counter machines [15], we can prove easily:

Lemma 2. *For every effective subset $S \subseteq \{0,1\}^{\mathbb{N}}$, there exists an oracle 4-counter machine that accepts exactly S.*

We will need this machine to be *reversible*. Informally, a DCM M is reversible (RCM) if there exists a DCM M' so that for every oracle w, $(s,c) \vdash (s',c')$ by M iff $(s',c') \vdash (s,c)$ by M', see [11] for a syntactical definition. Any DCM can be extended into a RCM by using two additional counters to store the previous instructions, so that we have

Lemma 3. *For every effective subset $S \subseteq \{0,1\}^{\mathbb{N}}$, there exists a oracle reversible 6-counter machine that accepts exactly S.*

Finally we add to this machine a new counter, that increases every two instructions. This ensures that every infinite computation of the counter machine, regardless of the first configuration, will never be periodic. We thus obtain an oracle reversible 7(!)-counter machine.

Note that it is quite likely that this can be encoded into a oracle 3-counter machine with the same properties, but there is no need for it (A 2-counter machine is unlikely, as one of the counters must be used somehow to track the position in the oracle.)

2.2 Turing machines

We now explain how the clever construction of Kari and Ollinger works, starting from an ordinary 2-counter machine. We will in the next section explain how to extend it for our purpose.

We begin by a generic simulation of a counter machine by Turing machines.

Let M be a 2-counter machine. Let $\Gamma = \{@, 0, x, y\}$ be a set of 4 different symbols.

A naive but effective way to encode these machines into Turing machines can be described as below: The state of the Turing machine will contain the state of the counter machine, and the tape will contain the word $@0^{c_1}x0^{c_2}y$ where c_i is the value of the i-th counter.

```
@0000000000000x0000000000000y
^
```

Now the behaviour of the Turing machine is quite clear: When it is at the position of $@$, it will scan the tape until x/y, deducing whether the i-th counter is zero, and changing it if necessary (which might need to shift the counters of indices greater than i), then coming back to $@$.

The main problem of this simulation is that it always has immortal uninteresting configurations, corresponding to unbounded searches for one of the

delimitor symbols: If the symbol is not present in the configuration, the search will be infinite.

To prevent this problem, the idea, originally from Hooper [8], is to use recursive calls. Suppose we are searching for the symbol a:

```
@000000000000x0000000000000y
 ^
 ?a
```

If the symbol a is not found in the next 3 cells of the tape, then we write `@`xy over the next 3 symbols of the tape, then *recursively* call the Turing machine.

```
@@xy000000000x0000000000000y
 ^
 s0
```

When (if) this nested simulation stops by reading the symbol a, the entire nested simulation is erased by doing it in *reverse*. We then can continue the search for a starting from the next symbol[1]

```
@000000000000x0000000000000y
  ^
  ?s
```

We refer the reader to [11] for more details. In particular we need to change the first symbol `@` into many different symbols to be able to keep into memory the state of the Turing machine before the recursive call.

An important feature of this construction is the following. If the counter machine has no periodic configuration (which happens as soon as one counter keeps increasing during the computation), then any computation of the Turing machine will contain computations of the counter machine starting from (almost) *anywhere*, in the following sense:

Proposition 2. *Let c be an immortal configuration of M. Then there exists an infinite interval I with the following property: for every $i \in I$, and every n so that $i+n \in I$, there exists a time during the computation of the Turing machine when the cells from i to $i+n$ contains the word `@`$xy0^{n-2}$, the head is in position i, and a recursive computation is started.*

[1] The construction in [11] actually starts the search from three symbols to the right (as we already know there is no symbol s in the next two symbols). It is clear that this does not change anything for their proof, but make Proposition 2 below true

2.3 The main construction

We now explain how the construction can be extended to work in our context. As explained above, we may see the Turing machine as having two tracks (but only one head): the first one has the original construction, and the other one a biinfinite word w over $\{0,1\}$. When the counter machine starts a computation from position i, it will try to accept the word u defined by $u_j = w_{i+j}$. Note that reading the letter of the oracle is quite easy: the position of the letter x is always where we want to read the oracle (that's why we chose the first counter to contain the position of the oracle)

Dealing with 7 counters instead of 2 requires no additional machinery, so we are nearly done.

However one problem remains. Suppose we start from a RCM recognizing the language $S \subseteq \{0,1\}^{\mathbb{N}}$. We now look at an infinite run of the Turing machine and we look at the interval I defined by Proposition 2. There are two cases for I:

- $I = [a, \infty[$ (possibly with $a = -\infty$) In this case, for all $i \geq a$, there exist arbitrary large computation starting from the cell i. This means that for all $i \geq a$, the word $(u_j)_{j \geq i} \in S$.
- $I = [-\infty, a[$. In this case, we can say nothing: we cannot find any position i for which we are certain that the word $(u_j)_{j \geq i}$ is in S.

We thus have to use a last additional trick to make the whole thing work: during the recursive call of the left searches, instead of using the same machinery, we will use another reversible counter machine, and run it *in the opposite direction*. That is, for the 2-counter machine, we will start e.g. from $YX@$ and the simulation will extend to the left, rather than the right.

Now this new construction starts from two reversible counter machines R_1 and R_2 and has the following property (the lemma is given for a oracle 2-counter machine. The change for a 7-counter machine is obvious):

Lemma 4. *Let c be an immortal configuration for the Turing Machine M. Then there exists an infinite interval $I = [a, b]$ (with $a = -\infty$ or $b = \infty$) with the following property*

- *for every $i \in I$, and every $n \geq 0$ so that $i + n \in I$, there exists a time during the computation of the Turing machine when the cells of the first track from i to $i + n$ contain the word $@xy0^{n-2}$, and a computation from R_1 is started from i.*
- *for every $i \in I$, and every $n \geq 0$ so that $i - n \in I$, there exists a time during the computation of the Turing machine when the cells from $i - n$ to i contain the word $0^{n-2}YX@$, and a computation from R_2 is started from i.*

We can now use all this refined construction to finally prove our main result:

Theorem 2 (main result). *For every effective set S, there exists a Turing machine M so that $I(M)$, the set of immortal configurations of M, is Muchnik equivalent to S.*

Proof. We start from S, and use theorem 3 to obtain a recursive language L so that the three subshifts defined by L are Muchnik equivalent to S. Now let R_1 and R_2 be two RCM that recognize respectively words with no factor in L and words with no factor in the mirror of L, and M be the Turing machine simulating R_1 and R_2 as above.

We now look at the immortal configurations for M.

- Let w be an biinfinite word avoiding L. Now it is clear that the configuration containing w on the second track and $@xy$ on the first track, with the head aligned with $@$, is an immortal configuration. Indeed, all factors of w (resp. of its mirror) are not in L (resp. its mirror), so that all computations of R_1 and R_2 will not halt. Hence, for every $w \in S(L)$, there exists an immortal configuration $c \in I(M)$ so that c is computable in w. In particular for every $u \in S$, there exists $c \in I(M)$ so that c is computable from u.
- Let x be a immortal configuration of the Turing machine, and I given by the lemma. Let u be the second track of x
  - If $I = [a, \infty[$, then the word w defined by $w_j = u_{a+j}$ avoids L, hence there exists a right-infinite word avoiding L that is Turing reducible to x. By Muchnik equivalence of $S^+(L)$ and S, there exists $u \in S$ that is computable from x.
  - If $I =]-\infty, b[$, then the word w defined by $w_j = u_{b-j}$ avoids the mirror of L, hence there exists a left-infinite word avoiding L that is Turing reducible to x. By Muchnik equivalence of $S^-(L)$ and S, there exists $u \in S$ that is computable from x.

  In both cases, we have found a word $u \in S$ that is computable from x.

□

It is important to note that we do not know, given a configuration x, whether the set I is left or right infinite, as we would need to simulate the Turing machine to do this. This means that the reduction from x to a right-infinite word is not uniform in x. As a side note, this means we have only proven a Muchnik equivalence and not a Medvedev equivalence.

3 Conclusion

We finish with a few applications. Using a well-known encoding of Turing machines into affine maps [5], we can prove:

Corollary 2 *There exists a piecewise affine map with rational coefficients and endpoints from* $[0,1]^2$ *to itself so that all computable points converge in finite time to* $(0,0)$*. However, there exists a noncomputable point that does not converge in finite time to* $(0,0)$*.*

Using the encoding of piecewise affine rational maps into tilings, we have an alternate proof of Myers' theorem [16]: There exists a tiling system that produces tilings, but no recursive tilings.

We end with an open question. For every right-computable real $\lambda > 0$, there exists an effective subshift of entropy λ [7]. Can we use our construction to prove that the entropy of a rational piecewise affine map can be any right-computable real λ ? This would extend a result of Koiran [12] proving that computing the entropy of a piecewise affine map is undecidable.

References

1. Vincent D. Blondel, Julien Cassaigne, and Codrin Nichitiu. On the presence of periodic configurations in Turing machines and in counter machines. *Theoretical Computer Science*, 289(1):573–590, 2002.
2. Douglas Cenzer, Ali Dashti, and Jonathan L. F. King. Computable symbolic dynamics. *Mathematical Logic Quarterly*, 54(5):460–469, 2008.
3. Douglas Cenzer and J.B. Remmel. Π^0_1 classes in mathematics. In *Handbook of Recursive Mathematics - Volume 2: Recursive Algebra, Analysis and Combinatorics*, volume 139 of *Studies in Logic and the Foundations of Mathematics*, chapter 13, pages 623–821. Elsevier, 1998.
4. Douglas Cenzer and Jeffrey Remmel. *Effectively Closed Sets.* ASL Lecture Notes in Logic, 2011. in preparation.
5. Pieter Collins and Jan H. van Schuppen. Observability of Hybrid Systems and Turing Machines. In *43rd IEEE conference on Decision and Control*, pages 7–12, 2004.
6. Jean-Charles Delvenne and Vincent D. Blondel. Quasi-periodic configurations and undecidable dynamics for tilings, infinite words and Turing machines. *Theoretical Computer Science*, 319:127–143, 2004.
7. Peter Hertling and Christoph Spandl. Shifts with Decidable Language and Non-Computable Entropy. *Discrete Mathematics and Theoretical Computer Science*, 10(3):75–94, 2008.
8. Philip K. Hooper. The Undecidability of the Turing Machine Immortality Problem. *Journal of Symbolic Logic*, 31(2):219–234, June 1966.
9. Emmanuel Jeandel and Pascal Vanier. Turing degrees of multidimensional SFTs. Submitted to Theoretical Computer Science, arXiv:1108.1012v2.
10. Carl G. Jockusch and Robert I. Soare. Degrees of members of Π^0_1 classes. *Pacific J. Math.*, 40(3):605–616, 1972.
11. Jarkko Kari and Nicolas Ollinger. Periodicity and Immortality in Reversible Computing. In *MFCS 2008*, pages 419–430, 2008.
12. Pascal Koiran. The Topological Entropy of Iterated Piecewise Affine Maps is Uncomputable. *Discrete Mathematics and Theoretical Computer Science*, 4(2):351–356, 2001.
13. Petr Kurka. On topological dynamics of Turing machines. *Theoretical Computer Science*, 174:203–216, 1997.
14. Joseph S. Miller. Two Notes on subshifts. *Proceedings of the American Mathematical Society*, 2011.
15. Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.
16. Dale Myers. Non Recursive Tilings of the Plane II. *Journal of Symbolic Logic*, 39(2):286–294, June 1974.
17. Stephen G. Simpson. Mass problems associated with effectively closed sets. *Tohoku Mathematical Journal*, 63(4):489–517, 2011.

18. Stephen G. Simpson. Medvedev Degrees of 2-Dimensional Subshifts of Finite Type. *Ergodic Theory and Dynamical Systems*, 2011.