From heaps of matches to the limits of computability

Urban Larsson and Johan Wästlund
Mathematical Sciences,
Chalmers University of Technology and University of Gothenburg,
Göteborg, Sweden
urban.larsson@chalmers.se, wastlund@chalmers.se

February 6, 2012

Abstract

We study so-called invariant games played with a fixed number d of heaps of matches. A game is described by a finite list $\mathcal{M}$ of integer vectors of length d specifying the legal moves. A move consists in changing the current game-state by adding one of the vectors in $\mathcal{M}$, provided all elements of the resulting vector are nonnegative. For instance, in a two-heap game, the vector $(1, -2)$ would mean adding one match to the first heap and removing two matches from the second heap. If $(1, -2) \in \mathcal{M}$, such a move would be permitted provided there are at least two matches in the second heap. Two players take turns, and a player unable to make a move loses. We show that these games embrace computational universality, and that therefore a number of basic questions about them are algorithmically undecidable. In particular, we prove that there is no algorithm that takes two games $\mathcal{M}$ and $\mathcal{M}'$ (with the same number of heaps) as input, and determines whether or not they are equivalent in the sense that every starting-position which is a first player win in one of the games is a first player win in the other.

1 Introduction

1.1 A children's game

A type of children's game for two players consists in placing a heap of matches on a table and taking turns removing them according to some simple rule, the winner being the person to make the last move. For instance, the rule can be that one is permitted to remove one, two or three matches. Playing a few games will lead to the insight that certain positions are advantageous in the sense that moving to them will secure the win. In this example, moving to a heap of four matches will secure the win in the next move. Similarly, a player moving to a heap of eight will be able to move to four in the next move, and in general, the *P-positions* are precisely the multiples of four.

If the set of numbers that one is allowed to remove from the heap is finite, then the set of P-positions will ultimately become periodic. Therefore any particular game of this type can be completely understood. Finding the period may require a tedious computation, but the game cannot in principle embrace any mysteries, see [BCG04].

1.2 Games of more than one heap

We study similar games with several heaps of matches. Each game has a fixed number d of heaps, and a position can be regarded as a d-dimensional vector $a = (a_1, \ldots, a_d)$ of non-negative integers. The rules of the game are encoded by a finite set $\mathcal{M}$ of integer vectors that specify the permitted moves. If $(m_1, \ldots, m_d) \in \mathcal{M}$, then from position $(a_1, \ldots, a_d)$, a player can move to position $(a_1 + m_1, \ldots, a_d + m_d)$, provided all of the numbers $a_1 + m_1, \ldots, a_d + m_d$ are nonnegative. These games are called *invariant* games [DR10, G66, L12, LHF11], since (apart from the nonnegativity condition) the move options given by $\mathcal{M}$ are independent of a. This is somehow implicit in the idea of heaps of matches — we should not have to count the remaining matches in order to play by the rules.

Since moves can involve adding matches to heaps, the game does not necessarily have to terminate. We restrict our attention to games where termination is not an issue by stipulating that for each $(m_1, \ldots, m_d) \in \mathcal{M}$, $m_1 + \cdots + m_d < 0$, so that in each move the total number of matches decreases.

Each game $\mathcal{M}$ has a set $P(\mathcal{M})$ of P-positions, and in informal terms our question is whether from the list $\mathcal{M}$ of permitted moves it is possible to

completely understand $P(\mathcal{M})$.

A first observation is that for any given position $a = (a_1, \ldots, a_d)$ we can determine recursively whether or not a is a P-position by first computing the status of all positions with fewer matches. The position a is in $P(\mathcal{M})$ iff there is no move from a to a position in $P(\mathcal{M})$. Consider for instance the game of two heaps given by $\mathcal{M} = \{(-1, -3), (-2, 1)\}$. The P-positions with fewer than 25 matches in each heap are shown in red in Figure 1.

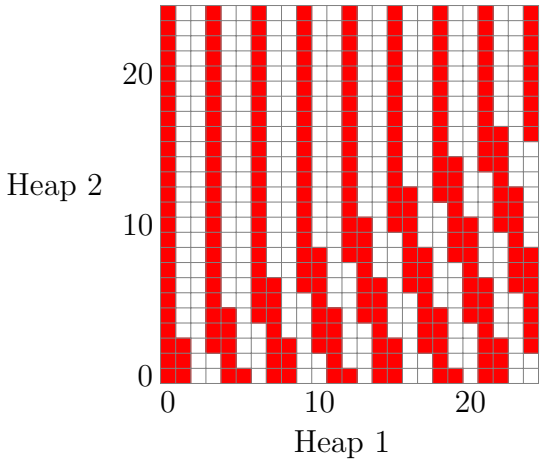

Figure 1: The P-positions of the game $\mathcal{M} = \{(-1, -3), (-2, 1)\}$.

There are two sectors, each with its own periodic pattern of P-positions, and clearly it would be possible to describe these patterns explicitly and prove by induction that they will persist. Therefore it is fair to say that we have complete understanding of the set of P-positions in this game.

For many games, the P-positions display some initial irregularities, after which they settle into a simple pattern. However, in some cases, the P-positions show no sign of regularity even after quite extensive computations. Figure 2 shows a typical example with large periodic regions interrupted by rather chaotic behavior along the borders of "conflicting" regions.

We claim that despite the apparent simplicity of the rules of these games, it is impossible in general to fully understand the behavior of the set $P(\mathcal{M})$ of P-positions from the list $\mathcal{M}$ of legal moves. It seems that such a claim requires a formal definition of "full understanding". To support our claim, we argue that full understanding, whatever it is, would at least imply the

ability to recognize when two games are *P-equivalent*, by which we mean that they have the same P-positions. We will prove that this is algorithmically undecidable.

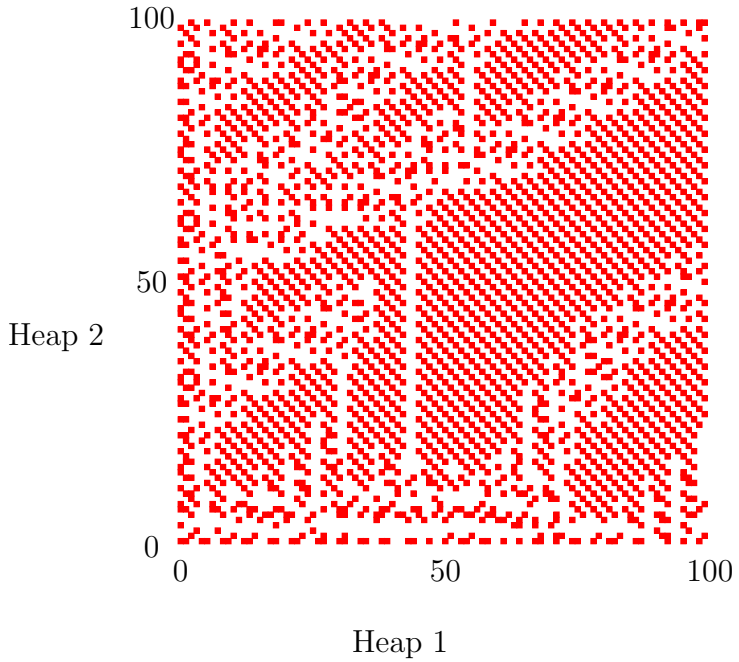

Figure 2: Initial P-positions of the game given by $\mathcal{M} = \{(0,-2),\ (-2,0),\ (2,-3),\ (-3,2),\ (-5,4),\ (-5,-2),\ (-4,-3),\ (-1,-4)\}$. Do they eventually settle into a pattern that can be fully understood, or is this a world of ever increasing complexity, where surprises will await us regardless of how far we take our computations?

Theorem 1. *There is no algorithm that, given as input the number d of heaps and two finite sets $\mathcal{M}$ and $\mathcal{M}'$ of integer vectors specifying the rules of two d-heap games, decides whether or not $P(\mathcal{M}) = P(\mathcal{M}')$.*

The rest of the paper is mainly devoted to the proof of Theorem 1. In Section 2 we describe a certain class of cellular automata for which several decision problems are known to be algorithmically unsolvable. In Section 3 we describe a class of (non-invariant) games called *modular games* that can emulate the cellular automata of Section 2. Finally in Section 4 we show

how invariant games can emulate modular games and how this establishes Theorem 1.

2 Cellular automata

Cellular automata (CAs) give rise to 2-dimensional patterns similar to that of Figure 1. For CAs it is known that some basic questions are algorithmically undecidable. Here we consider a restricted class of CAs. They have two states (0 and 1), and the state of cell i at time t is denoted by $x_{t,i}$. The starting configuration is ...000111..., or more precisely,

$$x_{0,i} = \begin{cases} 1, & \text{if } i \geq 0, \\ 0, & \text{if } i < 0. \end{cases}$$

The update rule is given by a number n and a Boolean function f taking n bits of input. The states are updated according to

$$x_{t+1,i} = f(x_{t,i-n+1}, x_{t,i-n+2}, \ldots, x_{t,i}).$$

In other words $x_{t+1,i}$ depends on $x_{t,i}$ and the $n-1$ cells immediately to the left of $x_{t,i}$. Moreover, for technical reasons, we require that $f(0, \ldots, 0) = 0$, which implies that $x_{t,i} = 0$ whenever $i < 0$.

We denote the cellular automaton corresponding to f by $CA(f)$.

Example

We demonstrate by taking $n = 2$ and letting the Boolean function be

$$f(x, y) = x \oplus y,$$

in other words, $f(x, y)$ is equal to 0 if $x = y$ and 1 if $x \neq y$. CAs are usually illustrated by drawing the tape from left to right, and for some reason it has become customary to let time flow downwards. Here we break this convention and draw time upwards. The 1's are represented by red squares. There is no need to include the positions $x_{t,i}$ for negative i, and therefore the leftmost column represents $x_{t,0}$.

Here the pattern is Pascal's triangle modulo 2, and although this pattern is non-periodic, it is still well understood. However, there are Boolean functions that give rise to more difficult behavior. The following result is well-known.

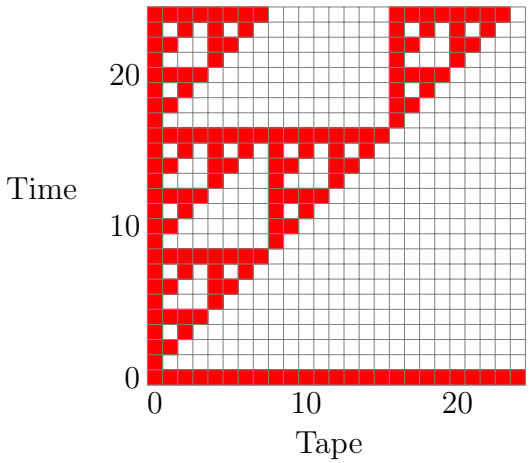

Figure 3: The cellular automaton given by $f(x, y) = x \oplus y$.

Lemma 2. *There is no algorithm that takes input an n-ary Boolean function f, and answers whether or not the string* 101 *ever occurs in CA(f).*

The reason that the lemma is true is essentially that a CA can emulate a generic Turing machine without the substring 101 occurring in the computation. The CA can then be "programmed" to print the string 101 if the emulated Turing machine halts, making the occurrence of 101 equivalent to the halting problem [T36].

The argument works also if the string 101 is replaced by 00 or 11, but 101 is the simplest string that will do for our subsequent purposes.

As the similarities between Figures 1 and 3 indicate, we will construct a game emulating a generic cellular automaton.

3 Modular games

We now describe a class of (non-invariant) games that we call *modular* games. We show that an arbitrary CA can be emulated by a modular game, and that the 101-occurrence problem can be reduced to P-equivalence of modular games.

A modular game $\mathcal{G}$ has only two heaps (which we call the tape- and the time-heap because they correspond to the tape- and time-axes in Figure 3).

For some positive integer k, there are finite sets $\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}$ of integer vectors that specify the available move options in $\mathcal{G}$. In a given position (a_1, a_2), where a_1 is the number of matches in the tape-heap and a_2 is the number of matches in the time-heap, the set of available moves is given by $\mathcal{M}_i$, where $0 \leq i < k$ and

$$i \equiv a_2 \pmod{k}. \tag{1}$$

Moreover we require that the number of matches in the time-heap decreases in every move and that the number of matches in the tape-heap does not increase. That is if $(m_1, m_2) \in \mathcal{M}_i$ then $m_1 \leq 0$ and $m_2 < 0$ and there is a move option from (a_1, a_2) to $(a_1 + m_1, a_2 + m_2)$ provided $a_1 + m_1 \geq 0, a_2 + m_2 \geq 0$ and (1) hold. We wish to prove the following lemma.

Lemma 3. *For each Boolean function f we can effectively construct two modular games $\mathcal{G}$ and $\mathcal{H}$ that are P-equivalent if and only if the word* 101 *never occurs in $CA(f)$.*

The first step is to construct a modular game that emulates $CA(f)$ for an arbitrary Boolean function f.

3.1 The modular games and computation

The number of matches in the tape-heap will correspond in the obvious way to a position on the tape. On the time axis we insert some space to allow for the rules of the modular game to "compute" the Boolean function f.

For some k which will have to depend on f, the positions with kt matches in the time-heap will correspond to the state of $CA(f)$ at time t. Hence the positions where the number of matches in the time-heap is divisible by k will encode the evolution of $CA(f)$.

Let square brackets $[\cdot]$ denote $1 - \max(\cdot)$, so that

$$[xyz] = 1 - \max(x, y, z),$$

etc, and by convention $[\,] = 1$. It is well-known that every Boolean function can be expressed in terms of nested brackets. This is because the set consisting of & and $\sim$ (negation) is a complete set of connectives in propositional logic. For instance, as can easily be checked, the function $f(x, y) = x \oplus y$ can be expressed as

$$x \oplus y = [[xy][[x][y]]].$$

If we let P-positions correspond to the value 1 and N-positions (the non-P positions) to the value 0, then the value of a position can be recursively computed by the bracket-function. If a position has moves to positions of values $x_1, \ldots, x_k$, then its value is $[x_1 \cdots x_k]$.

Therefore we can construct a game that computes the function f by letting intermediate positions have values $[x]$, $[y]$, $[xy]$ and $[[x][y]]$. Figure 4 illustrates how this is done for $k = 5$.

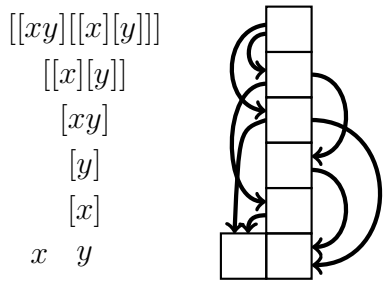

Figure 4: A modular game computing $f(x, y) = x \oplus y$ in five steps. The arrows indicate move options. The value of each cell is the bracket of the values of all its options.

The construction in Figure 4 corresponds to

$$\begin{aligned}
\mathcal{M}_0 &= \{(0,-1),(0,-2)\} \\
\mathcal{M}_1 &= \{(-1,-1)\} \\
\mathcal{M}_2 &= \{(0,-2)\} \\
\mathcal{M}_3 &= \{(0,-3),(-1,-3)\} \\
\mathcal{M}_4 &= \{(0,-2),(0,-3)\}
\end{aligned}$$

Figure 5 shows how this modular game emulates the cellular automaton given by $f(x, y) = x \oplus y$. Every fifth row corresponds to a row of Figure 3.

3.2 The check for 101

The second step concerns the check for the word 101. Given a modular game $\mathcal{G}$, corresponding to a Boolean function f as illustrated in Figure 4, we can construct a modified game where, inside the computation of f, we have built in a check of whether or not the input to f ends with the substring 101.

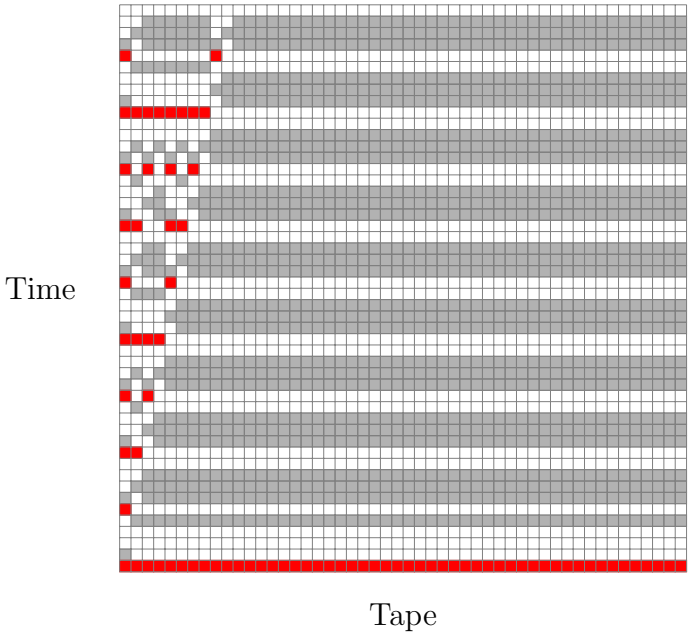

Figure 5: A modular game emulating $f(x, y) = x \oplus y$. Here the P-positions with fewer than 50 matches in each heap are represented by filled squares. Rows corresponding to $a_2 \equiv 0 \pmod 5$ are highlighted by drawing the P-positions in red.

The check takes place before the actual computation of f. First we introduce boxes b_1 and b_2 that invert the last position and the third position from the end of the input (which is where we want to check if there are 1's in order to check if the input ends by 101). Then the box b_3 that actually checks for 101 is connected to the penultimate position in the input, and to b_1 and b_2. We denote this game by $\mathcal{G}'$.

We also construct a "dummy" game $\mathcal{G}''$ that looks like $\mathcal{G}'$ but doesn't check for 101. In $\mathcal{G}''$, the box b_3 is instead connected only to b_2 and to the third box from the end of the input. Therefore the box b_3 will always be an N-position (a zero) in $\mathcal{G}''$, while in $\mathcal{G}'$ it will be a P-position whenever the input string ends by 101. Figures 6 and 7 depict the construction of the games $\mathcal{G}'$ and $\mathcal{G}''$.

Hence, the modular games $\mathcal{G}'$ and $\mathcal{G}''$ are P-equivalent if and only if the string 101 never occurs in $CA(f)$. This proves Lemma 3.

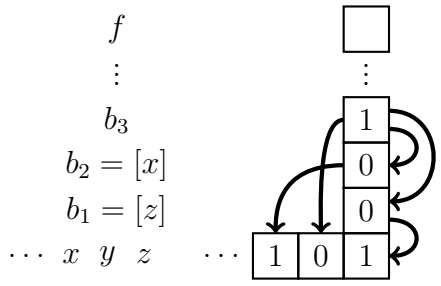

Figure 6: A modular game $\mathcal{G}'$ with input xyz computing a function f in a few steps. (We have omitted the moves that actually compute f and any remaining variables.) The first two boxes b_1 and b_2 invert x and z. The third box b_3 is a 1 (=P-position) if and only if $xyz = 101$.

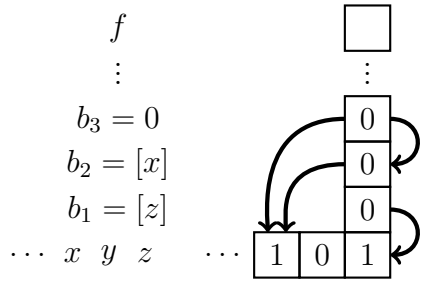

Figure 7: A modular game $\mathcal{G}''$ whose output function f is identical to that of $\mathcal{G}'$. The only difference is that $\mathcal{G}''$ does not check for the pattern 101. The box b_3 contains a 0 independently of the input xyz to f.

4 Emulating a modular game by an invariant game

The following lemma, which we prove in this section, provides the remaining link to Theorem 1.

Lemma 4. *For two modular games $\mathcal{G}$ and $\mathcal{H}$, we can effectively construct invariant games $\mathcal{M}$ and $\mathcal{M}'$ that are P-equivalent if and only if $\mathcal{G}$ and $\mathcal{H}$ are P-equivalent.*

We introduce a gadget that allows us to emulate a modular game by an invariant game. The resulting invariant game has a time-heap and a tape-

heap, and k more heaps that we call the *gadget*, keeping track of the size of the time-heap modulo k. The gadget consists of k heaps where the positions that we are primarily interested in are those where one of the heaps contains a match and the remaining $k-1$ heaps are empty (we return to what happens if this is not the case). The moves are designed so that the match in the gadget follows the time-heap modulo k. The heaps of the gadget may be numbered $0, \ldots, k-1$, and for each of the move sets $\mathcal{M}_i$ of the modular game, we introduce corresponding moves in the invariant game where the ith heap of the gadget is emptied, the tape- and time-heaps are affected as given by some move in $\mathcal{M}_i$, and a match is added to the heap of the gadget corresponding to the new modulus of the time-heap.

There is a small technicality to address here. In order for the construction to work, we must require that each move in the modular game changes the congruence class of the number of matches in the time-heap. But this constraint is taken care of by replacing k by a multiple of k which is larger than the number of matches removed from the time-heap in any of the moves (a k-modular game can be described as an lk-modular game for every positive integer l).

Clearly there is a subset of positions of the invariant game that emulate the modular game, namely those where (i) there is exactly one match in the gadget, and (ii) this match is in the heap corresponding to the number of matches in the time-heap modulo k.

4.1 Reducing pattern-occurrence to P-equivalence

In order to conclude that the question of P-equivalence of invariant games is algorithmically undecidable, we wish to reduce the pattern occurrence problem of a generic CA to P-equivalence. We did this for modular games in Section 3. It remains to check that the result actually carries over to invariant games through the gadget-trick.

4.1.1 If the gadget is out of phase

In case a single match in the gadget is out of phase with the congruence class of the time-heap, there is a first row $0 < i < k$ which corresponds to a finished computation as in Figure 4. The gadget treats this row as if it were congruent to 0 modulo k (that is permitting moves as defined by $\mathcal{M}_0$). But, since $i - k < 0$, by the convention $[\,] = 1$, $f(x_1, \ldots, x_n)$ is defined uniquely

by its empty input producing an output which is independent of the position of the tape-heap. Hence, the information in row i must be constant. If it is constant 0 (N-positions), then the following pattern will be periodic, since the computation now restarts as if it had started on a tape of only zeros. If on the other hand it is constant 1 (P-positions), then the behavior from row i and onwards will be the same as if the gadget was in phase, since the computation now starts from a row identical to that of the starting configuration of the CA, that is $\cdots 000111 \cdots$.

4.1.2 If the gadget contains more than one match

What happens in positions where the gadget contains more than one match? Say that there is a match in heap i of the gadget and another in heap $j \neq i$. Then a legal move in the time-heap can be obtained from either $\mathcal{M}_i$ or $\mathcal{M}_j$ independent of its number of matches, so that the rules of the invariant game will not encode the evolution of $CA(f)$ as prescribed by the modular games in Section 3. Hence we would like to make positions with more than one match in the gadget trivial. We therefore choose some large number N, and allow any move that transfers two matches in the gadget to two other heaps, and removes any number smaller than N from the two main heaps. Since the number of matches in the gadget will never change, this will give a trivial periodic pattern of P-positions in the main heaps.

5 Conclusion and questions

In conclusion, a sub-class of all invariant games emulate the modular games, for which, by Lemma 2 and Lemma 3, it is undecidable whether or not two games are P-equivalent. Altogether, this proves Lemma 4 and hence also Theorem 1. As a consequence of our approach it is also algorithmically undecidable whether a given finite pattern occurs in the set of P-positions of an invariant game.

How many heaps are required for undecidability? (Strictly speaking we didn't prove that *any* finite number of heaps leads to undecidability). We guess that three heaps suffice, and perhaps even two since it is easy to generate complicated patterns of P-positions as in Figure 2 with small finite sets of moves.

Do we need to be able to add matches to heaps in order to achieve undecidability?

If there are no restrictions on moves, so that the total number of matches may increase, is the outcome (P, N or draw) of a specific position decidable?

References

[BCG04] E. R. Berlekamp, J. H. Conway, R. K. Guy, *Winning ways*, **1-2** Academic Press, London (1982). Second edition, **1-4**. A. K. Peters, Wellesley/MA (2001/03/03/04).

[DR10] E. Duchêne and M. Rigo, Invariant Games, *Theoret. Comput. Sci.*, Vol. 411, 34-36 (2010), pp. 3169–3180

[G66] S. W. Golomb, A mathematical investigation of games of "take-away". *J. Combinatorial Theory* **1** (1966) pp. 443–458.

[L12] U. Larsson, The $\star$-operator and invariant subtraction games, *Theoret. Comput. Sci.*, Vol. 422, (2012) pp. 52–58.

[LHF11] U. Larsson, P. Hegarty, A. S. Fraenkel, Invariant and dual subtraction games resolving the Duchêne-Rigo Conjecture, *Theoret. Comp. Sci.* Vol. 412, 8-10 (2011) pp. 729–735.

[T36] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. London Math. Soc., series 2*, **42** (1936-37), pp. 230–265.

[W02] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., (2002).