A Note on Culik-Yu Classes

KLAUS SUTNER
Stevens Institute of Technology
Hoboken, NJ 07030

Abstract

Culik and Yu suggested a classification of cellular automata into four classes based on Wolfram's earlier heuristic classification. The purpose of this note is to determine the position of these classes within the arithmetical hierarchy. We will show that CLASS ONE and CLASS TWO are Π^0_2-complete whereas CLASS THREE is Σ^0_3-complete. CLASS FOUR is trivial.

1 Introduction

In [6] Wolfram gave a heuristic classification of cellular automata into four types. His classification is based on the evolution of configurations and uses easily observable characteristics of the behavior of the cellular automaton. In a recent paper by Culik and Yu [1] the authors formalize Wolfram's classification. Again, four types of cellular automata are considered. The corresponding classes will be denoted CLASS ONE, CLASS TWO, CLASS THREE and CLASS FOUR. Informally, they are defined as follows. Let ρ be the transition rule of a cellular automaton.

1. Rule ρ is in CLASS ONE iff every finite configuration evolves to a stable configuration in finitely many steps under rule ρ.

2. Rule ρ is in Class Two iff every finite configuration evolves to a periodic configuration in finitely many steps under rule ρ.

3. Rule ρ is in Class Three iff it is decidable whether a configuration occurs in the orbit of another.

4. Class Four comprises all local rules.

Note that the classes form a hierarchy. It is shown in [1] that it is undecidable to which class a given cellular automaton belongs. More precisely, their arguments show that:

(a) Class One and Class Two are Π^0_1-hard.
(b) Class Three is Σ^0_1-hard.

By counting quantifiers, one can see from the definitions that Class One and Class Two lie at level Π^0_2 and Class Three at level Σ^0_3 in the arithmetical hierarchy. Thus there is a significant gap between the lower bounds established by Culik and Yu and the obvious upper bounds. We will close this gap and show that in fact:

(a') Class One and Class Two are Π^0_2-complete.
(b') Class Three is Σ^0_3-complete.

Thus Class One and Class Two are equidecidable with the problem of deciding whether a recursively enumerable set is infinite. Similarly Class Three is equidecidable with the problem of deciding whether a recursively enumerable set is recursive.

Our arguments are based on a technical lemma which shows that the Gödel numbers of Turing machines that halt on all configurations is Π^0_2-complete. This lemma also provides a proof for corollary 2 in [1] that does not use cellular automata.

To keep this note reasonably short we refrain from repeating the basic definitions of recursion theory. In particular we will not introduce the arithmetical hierarchy and refer the reader to [2] or [3] for definitions of the classes Π^0_2, Σ^0_3 and so forth. The necessary definitions for cellular automata as well as for Turing machines are presented briefly in the next section. Section 3 contains proofs for the completeness results.

2 Definitions

We will consider exclusively one-dimensional cellular automata. Every cell can assume a finite number of possible states; the collection Σ of possible states is called the *alphabet* of the automaton. A map $X : \mathbf{Z} \rightarrow \Sigma$ from the set of all cells to the alphabet is a *configuration* of the cellular automaton. $\mathcal{C}$ denotes the space of all configurations. A *local rule* is a map $\rho : \Sigma^N \rightarrow \Sigma$ where $N \subset \mathbf{Z}$ is a finite set, called the *basic neighborhood* of the rule. The rule ρ is extended to a *global rule* (also denoted by ρ) $\rho : \mathcal{C} \rightarrow \mathcal{C}$ as follows. Given a configuration X define for any cell c the *local configuration* at c, $X_c : N \rightarrow \Sigma$, by $X_c(z) := X(c+z)$. Then $\rho(X)(c) := \rho(X_c)$.

For a state s in Σ let Z_s be the local configuration defined by $Z_s(c) = s$ for all c in N. Similarly X_s is the global configuration defined by $X_s(c) = s$ for all c in Z. State s in Σ is *stable* iff $\rho(Z_s) = s$. Suppose s is stable. The *s-support* of a configuration X is the collection of cells $\{c \mid X(c) \neq s\}$. The configuration X is *s-finite* iff its s-support is finite. A configuration X is *periodic* iff for some $t < 0$: $\rho^t(X) = X$. The *orbit* of X is the collection of all configurations $\rho^t(X), t \geq 0$.

We can now give a precise definition of the Culik-Yu classes.

The Classification

1. ρ is in Class One iff there exists a stable state s in Σ such that every s-finite configuration evolves to X_s in finitely many steps under rule ρ.

2. ρ is in Class Two iff there exists a stable state s in Σ such that every s-finite configuration evolves to a periodic configuration in finitely many steps under rule ρ.

3. ρ is in Class Three iff it is decidable whether a configuration occurs in the orbit of another.

Coding

We briefly indicate how to code rules, configurations and so forth as non-negative integers. It is convenient to assume that $\Sigma = \{0, 1, \ldots, k-1\}$ for some $k \geq 2$. Let ρ be an arbitrary local rule. To code ρ, we order the local configurations lexicographically as $Z_1, .., Z_{k^n}$ where $n := |N|$. Then we code ρ by

$$C(\rho) := \langle \rho(Z_1), \ldots, \rho(Z_{k^n}) \rangle.$$

Here $\langle .. \rangle$ is any standard coding function, see e.g. [3]. Let Σ_0 be the set of stable states and s a symbol in Σ_0. For a s-finite configuration X let $C(X)$ denote its code number. Also let $Con_{s,e}$ be the codes of all s-finite configurations for rule ρ_e. $Con_{s,e}$ is primitive recursive uniformly in e and s. It is straightforward to show that there is a primitive recursive predicate Succ such that for all s-finite configurations X and Y we have

$$\rho^t(X) = Y \text{ iff } \mathrm{Succ}(t, C(X), C(Y), C(\rho)).$$

Thus a configuration Y occurs in the orbit of another configuration X under rule ρ iff

$\exists t \mathrm{Succ}(t, C(X), C(Y), C(\rho))$. This shows that orbits are r.e. One can easily construct rules with non-recursive orbits.

Turing Machines

Our completeness proofs below are all based on simulations of Turing machines on cellular automata. It will be convenient for our purposes to define a *Turing machine* as a quintuple $M = \langle Q, \Gamma, \delta, q_0, q_H \rangle$. Here Q is a finite set of states, $q_0 \in Q$ is the initial state and $q_H \in Q$ is the halting state. Γ is the tape alphabet of the machine. The partial map $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function of M. We assume that δ is defined everywhere except on arguments $(q_H, \sigma), \sigma \in \Gamma$. Thus M halts if and only if state q_H is reached at some point during the computation. We will always require M to erase its tape before it halts. As is customary an *instantaneous description* (ID) of M is a word in $\Gamma^* Q \Gamma^*$ (where we assume that Γ and Q are disjoint). We may assign a Gödel number e to every Turing machine. This provides a standard enumeration $(M_e)_{e \geq 0}$ of all Turing machines. Notice that there are primitive recursive predicates ID_e and $\vdash_e^\sigma$ (uniformly in e and σ) such that I is in ID_e iff I codes an ID of M_e and $x \vdash_e^\sigma y$ iff $x, y \in ID_e$ and Turing machine e moves from the ID coded by x to the ID coded by y in σ steps.

For any number $x \geq 0$ we write I_x for the initial ID on machine M_e corresponding to input x. $M_e(x) \downarrow$ denotes the fact that M_e on I_x halts after finitely many steps:

$$\exists \sigma (I_x \vdash_e^\sigma q_H).$$

Similarly we write $M_e(x) \uparrow$ if M_e fails to halt on x. For I in ID_e we write $M_e[I] \downarrow$ (respectively, $M_e[I] \uparrow$) iff M_e started on ID I halts after finitely many steps (respectively, fails to do so). In the future we will not distinguish between IDs and their codes and write for example $UqV \vdash_e^1 U'q'V'$ to indicate that Turing machine number e moves from UqV to $U'q'V'$ in one step.

As usual let $W_e := \{x \mid M_e(x) \downarrow\}$ be the e-th r.e. set. For our completeness arguments we will need the set of (Gödel numbers of) all Turing machines that halt on all inputs and that halt on a decidable set of inputs, respectively. More precisely define

$$TOT := \{e \mid W_e = \mathbf{N}\}$$

and

$$REC := \{e \mid W_e \text{ is recursive }\}.$$

It is well known that TOT is Π_2^0-complete and REC is Σ_3^0-complete, see, e.g., [2].

The major technical obstacle in the simulation of a Turing machine on a cellular automaton is the following: in recursion theory one is only interested in computations of Turing machines, i.e., sequences of IDs that start with an initial ID I_x for some x. In the context of cellular automata, however, one has to contend with the orbits of arbitrary configurations. Due to the fact that as a set of words ID_e is regular, it is not difficult to eliminate configurations that do not correspond to any ID whatsoever. In fact, given the proper coding, a cellular automaton can detect non-IDs in one step. A more serious problem is caused by IDs that do not occur during any computation of M_e. We call such IDs *inaccessible*. Thus I in ID_e is inaccessible iff

$$\neg \exists x, \sigma (I_x \vdash_e^\sigma I).$$

The set of inaccessible IDs of machine M_e is therefore co-r.e. (or Π_1^0 in the arithmetical hierarchy). In fact, it is not hard to see that this set is in general Π_1^0-complete.

It may well happen that for some index e in TOT the machine M_e fails to converge on some inaccessible ID I. Define ALL to be the set of Gödel numbers of Turing machines that halt on all IDs. Observe that ALL, unlike TOT, fails to be an index set. In a lemma below we will show that there is a primitive recursive function p such that for all $e \geq 0$, machines M_e and $M_{p(e)}$ accept the same inputs and $M_{p(e)}$ halts on all its inaccessible IDs.

Upper Bounds

As is customary with decision problems, Class One may be construed as a set of natural numbers: rule ρ is in Class One iff $C(e) \in$ Class One. It follows that Class One is Π^0_2:

$$\text{r} \in \text{Class One iff } \exists s \in \Sigma_0 \forall x \in Con_{s,r} \exists t (\text{Succ}(t, x, X_s, r)).$$

Similarly Class Two is in Π^0_2: r $\in$ Class Two iff

$$\begin{gathered}\exists s \in \Sigma_0 \forall x \in Con_{s,r} \exists 0 \leq t < t', y \in Con_{s,r} \\ (t < t' \wedge \text{Succ}(t, x, y, r) \wedge \text{Succ}(t', x, y, r)).\end{gathered}$$

For Class Three we have r $\in$ Class Three iff

$$\exists e \in REC \forall x, y \in Con_r (\exists t \text{Succ}(t, x, y, r) \Leftrightarrow (x, y) \in W_e).$$

Since REC is Π^0_2, Class Three is Σ^0_3.

3 Completeness Results

Throughout this section we will assume that state 0 is stable. The 0-finite configurations are referred to simply as finite configurations. Our first step is to show that TOT and ALL are recursively isomorphic. By Myhill's theorem we only have to show that TOT is one-one reducible to ALL and vice versa. The next proposition contains the easy direction ALL $\leq_1$ TOT.

Proposition 3.1 *ALL is one-one reducible to TOT.*

Proof. There is a primitive recursive function f such that

$$M_{f(e)}(x) = \begin{cases} 0 & \forall z \in ID_e, z < x (M_e[z] \downarrow), \\ \uparrow & \text{otherwise.} \end{cases}$$

Hence $f(e) \in$ TOT iff M_e converges on all its configurations iff $e \in$ ALL. One can easily make sure that f is injective. □

Lemma 3.1 *There is an injective primitive recursive function p such that for all $e \geq 0$:*
(1) $W_e = W_{p(e)}$
(2) $M_{p(e)}$ *halts on all its inaccessible IDs.*

Proof. We will show that for any Turing machine M_e there exists a modified machine $M_{e'}$ such that
- for any $x \geq 0 : M_e(x) \downarrow \iff M_{e'}(x) \downarrow$, and
- for any inaccessible ID I of $M_{e'}$: $M_{e'}[I] \downarrow$.

Moreover, it will be clear from the construction that the index e' can be computed primitive recursively from e. As pointed out in the introduction, the class of inaccessible IDs is in general Π^0_1-complete. Thus we cannot effectively eliminate these IDs. We consider instead IDs of M_e with an additional tag: the tag contains two numbers x and σ such that - supposedly - the ID in question occurs after σ steps in the computation of M_e on input x. A tagged ID (x, σ, I) is *correct* iff indeed $I_x \vdash^{\sigma}_{e} I$. Unlike accessibility correctness of tagged IDs is a primitive recursive property and can thus be verified by the Turing machine $M_{e'}$. If the tag is correct, machine $M_{e'}$ will generate the next ID of M_e. Also, the counter on the tag will be changed changed to $\sigma + 1$ to preserve correctness. Otherwise the machine $M_{e'}$ halts. The process then starts anew.

More precisely, the Turing machine $M_{e'}$ functions as follows. Starting at an initial ID $I_x = q_0 1^x$, machine $M_{e'}$ first changes the tape inscription to

$$\#1^x\#\#q'_0 1^x\#$$

where q'_0 is the initial state of M_e. This new configuration of $M_{e'}$ represents the tagged ID $(x, 0, I_x)$ of M_e. Machine $M_{e'}$ then cycles through the following three phases.

Tape Verification

During this phase $M_{e'}$ tests whether its tape contains a tagged ID of M_e, i.e., an inscription of the form

$$\#1^x\#1^\sigma\#UqV\# \qquad (1)$$

where $x, \sigma \geq 0, U, V \in \Gamma^*$ and $q \in Q$. If the verification fails $M_{e'}$ halts.

Accessibility Test

In this phase $M_{e'}$ will check whether the tagged ID on its tape is correct. The portion of the tape used during the test are subsequently erased, so the tape inscription will be back to (1) after successful completion of the test. Failure will cause $M_{e'}$ to halt.

Next ID

We may now assume that the tape contains a correctly tagged ID (x, σ, I). The machine now determines whether I is the halting configuration of M_e. If so, $M_{e'}$ also halts. Otherwise $M_{e'}$ computes the next ID and replaces the old ID on its tape by the new one. Furthermore it increments the counter σ to $\sigma + 1$.

This completes the definition of $M_{e'}$.

Since on any input $x \geq 0$ machine $M_{e'}$ simply simulates machine M_e, albeit in a very circuitous fashion, we have that $M_{e'}$ halts on x iff M_e halts on x. Hence $W_{e'} = W_e$ and it remains to show that $M_{e'}$ halts on all its inaccessible configurations. So suppose $M_{e'}$ is started on an arbitrary ID and performs an infinite computation. The crucial observation is that that $M_{e'}$ can perform only finitely many moves before it must enter a tape verification phase (recall that IDs are finitary objects). As $M_{e'}$ never halts the verification must be successful, hence the tape inscription must have the form $\#1^x\#1^\sigma\#I\#$ and represents a tagged ID (x, σ, I) of M_e. Next $M_{e'}$ tests accessibility of I (for machine M_e). Since no failure occurs we must have $I_x \vdash_e^\sigma I$. But then $\#1^x\#1^\sigma\#I\#$ is also accessible for $M_{e'}$:

$$I_x = q_0 1^x \vdash_{e'}^\tau \#1^x\#1^\sigma\#I\#$$

for some number $\tau \geq \sigma$. Hence machine $M_{e'}$ cannot perform an infinite computation on an inaccessible ID and we are done. $\Box$

Corollary 3.1 *ALL, the set of Gödel numbers of Turing machines that halt on all configurations, is* Π_2^0*-complete.*

Proof. With the preceding definitions we have

$$e \in \mathrm{ALL} \quad \text{iff } \forall I \in ID_e(M_e[I] \downarrow) \quad \text{iff } \forall I \in ID_e \exists \sigma (I \vdash_e^\sigma q_H).$$

Hence ALL is Π^0_2. By the lemma $e \in$ TOT iff $p(e) \in$ ALL. Hence TOT $\leq_1$ ALL which shows that ALL is Π^0_2-complete. $\Box$

With proposition 3.1 we can conclude that TOT and ALL have the same one-one degree and are therefore recursively isomorphic. We can now localize Class One, Class Two and Class Three within the arithmetical hierarchy.

Theorem 3.1 Class One *is* Π^0_2*-complete.*

Proof. By corollary 3.1 we only have to show that ALL, the set of Gödel numbers of Turing machines that halt on all configurations, is reducible to Class One. To this end we will construct a rule ρ_e for every e such that e is in ALL iff ρ_e is in Class One. The construction of rule ρ_e is rather standard, we therefore will omit any details and only give a brief description. ρ_e first tests whether configuration X corresponds to an ID of M_e (more precisely, whether every isolated non-quiescent part of X corresponds to an ID; there may several such parts). This is possible in one step if one augments the tape alphabet of M_e by indicator bits that determine the position of the head relative to the symbol. If the test fails the quiescent configuration is generated in $O(n)$ steps where n is the number of cells in the support of X. Otherwise ρ_e simulates the computation of M_e on this ID. If M_e ever halts the quiescent configuration is generated, otherwise no stable configuration occurs. Again the map $e \mapsto e'$ is clearly primitive recursive and also injective. Hence ALL $\leq_1$ Class One and we are through. $\Box$

Combining the lemma and the technique of the last theorem we obtain a lower bound for Class Two and Class Three as follows.

Theorem 3.2 Class Two *is* Π^0_2*-complete.*

Proof. We claim that $e \in$ TOT iff $\rho_{p(e)}$ is in Class Two. Here p is the primitive recursive function of the lemma and ρ_i is the rule constructed in theorem 3.1. To see this first suppose e is in TOT. Then $M_{p(e)}$ halts an all its IDs, whence every configuration evolves to the stable configuration X_0 under rule $\rho_{p(e)}$. On the other hand assume e is not in TOT and pick some x such that $M_e(x) \uparrow$. Consider the orbit of configuration

$$X = \#x\#0\#I_x\#$$

under rule $\rho_{p(e)}$. It clearly contains configurations of the form $\#x\#\sigma\#I\#$ for all $\sigma \geq 0$. Hence the orbit of X fails to be periodic and rule $\rho_{p(e)}$ is not in Class Three. □

Theorem 3.3 Class Three *is* Σ^0_3*-complete.*

Proof. Again let p be the primitive recursive function of the lemma and ρ_i the rule constructed in theorem 3.1. Recall that REC is the collection of indices of recursive r.e. sets. We claim that $e \in$ REC iff $\rho_{p(e)}$ is in Class Three.

To see this first suppose W_e is recursive and let X and Y be two arbitrary finite configurations of rule $\rho_{p(e)}$. Note that it is decidable whether X corresponds to an accessible ID of $M_{p(e)}$. If not, the orbit of X must be finite and we can test whether Y occurs in the orbit by enumerating it. If X is accessible, say, from initial ID I_x, we first test whether x is in W_e. This can be done effectively since W_e is recursive. If indeed x lies in W_e the orbit of X must again be finite and we can test whether Y occurs in it by enumerating it. Otherwise the orbit of X is infinite but the configurations that occur in it are essentially all of the form $\#1^x\#1^\tau\#I\#$. Hence it is easily decidable whether Y lies in the orbit of X: we have to execute at most τ cycles in the computation of $M_{p(e)}$.

For the opposite direction note that

$$x \in W_e \quad \text{iff} \quad M_e[I_x]\downarrow \quad \text{iff} \quad M_{p(e)}[I_x]\downarrow \quad \text{iff} \quad \exists t(\rho^t_{p(e)}(q_0 1^x) = X_0).$$

Thus x is in W_e iff the orbit of I_x contains configuration X_0. Hence W_e is decidable whenever $\rho_{p(e)}$ is in Class Three. □

4 Conclusion

We have shown that the natural hierarchy of cellular automata proposed by Culik and Yu is computationally highly unfeasible. Not only are the classes undecidable (i.e., they are not located at level Δ^0_1 in the arithmetical hierarchy), they are in fact recursively isomorphic to the index sets TOT

(Class One and Class Two) and REC (Class Three) respectively. TOT is complete for level Π^0_2 and REC is complete for level Σ^0_3 of the arithmetical hierarchy. Hence a powerful ad hoc argument is required to determine the class of any specific local rule. It is hardly surprising that even for some totalistic rules of width 3 their classification is not known, see e.g. [5].

It is interesting to note that a classification of finite cellular automata similar to the Culik-Yu hierarchy meets with certain difficulties at level three. The problem to determine whether a configuration occurs in the orbit of another is in general **PSPACE**-complete even for one-dimensional finite cellular automata. In analogy to Class Three call a rule predictable iff one can solve this problem in polynomial time. The existence of a rule that fails to be predictable is then equivalent to the assertion that **P**$\neq$ **PSPACE**, one of the more notorious open problems of complexity theory. Hence one might expect it to be rather difficult to characterize the class of all predictable rules. See [4] for a number of results on the complexity of various decision problems associated with the evolution of configurations on finite cellular automata.

Acknowledgements

It is a pleasure to acknowledge many helpful discussions with my colleague S.L.Bloom.

References

[1] K. Culik II and Sheng Yu. Undecidability of CA classification schemes. *Complex Systems*, 2(2):177–190, 1988.

[2] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw Hill, 1967.

[3] J. R. Shoenfield. *Mathematical Logic.* Addison Wesley, 1967.

[4] K. Sutner. The complexity of finite cellular automata. Submitted.

[5] S.Wolfram. Computer software in science and mathematics. *Scientific American*, 251(3):188–203, 1984.

[6] S.Wolfram. Universality and complexity in cellular automata. *Physica 10D*, pages 1–35, 1984.