Computing Power of the Game *Manufactoria*

Yanheng Wang

January 6, 2021

Abstract

Manufactoria is a game that asks players to build a factory pipeline for quality control. This short note formalises the computational model underlying the game and discusses its computing power. We show that the model is Turing complete, and that the time and space complexity coincide in this model.

1 The Factory Model

The factory allows several sorts of facilities:

Facility		**Function**
empty	□	rejects the product
conveyor	▲	moves the product to the next cell pointed by the arrow
readers	◆ ◆	tries to read out the first dot on product's tape, then branches according to its colour (black = unmatched)
printers	💧 💧 💧 💧	appends a dot on product's tape, with specified colour

We let $F := \{\square, ▲, ◆, ◆, 💧, 💧, 💧, 💧\}$ be the set of facilities. The alphabet of the product tape is $\Sigma := \{B, R, Y, G\}$. A factory can thus be abstracted as an $m \times m$ square matrix A. Each element A_{ij} is a 3-tuple (f, o, s) where $f \in F$ is the facility of the (i, j)-cell, $o \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ is its orientation ($\uparrow$ being the upright position) and $s \in \{0, 1\}$ specifies whether we swap colours of its two wings.

During the execution of our factory model, the state at any given time can be encoded by a *configuration* $(i, j, x) \in [m] \times [m] \times \Sigma^*$, where i, j specifies the position of the product, and x records its tape content.

We are now ready to define the *yielding relation* $(i, j, x) \implies (i', j', x')$, which is almost self-evident. To avoid the unpleasant prolixity, we only mention some typical cases. Let $(f, o, s) := A_{ij}$.

1. If $f = \square$, then $(i, j, x) \implies (i', j', x')$ iff $(i, j, x) = (i', j', x')$.

2. If $f =$ ◆ and $x = Rw$ where $w \in \Sigma^*$, then $(i, j, x) \implies (i', j', x')$ iff $x' = w$ and (i', j') is one step away from (i, j) to the direction of the red arrow.

3. If $f =$ ◆, $x = wa$ where $w \in \Sigma^*$ and $a \notin \{B, R\}$, then $(i, j, x) \implies (i', j', x')$ iff $x = x'$ and (i', j') is one step away from (i, j) to the direction o. In words, the product is passed to the black arrow and x remains unmodified in the case that the leading colour on tape fails to match that of the reader's.

4. If $f =$ 💧, then $(i, j, x) \implies (i', j', x')$ iff $x' = xB$ and (i', j') is one step away from (i, j) to the direction o.

Note that our definition naturally rules out the possibility of "running away from the factory"; in that case, the yielding relation would get stuck since $i', j' \in [m]$.

Definition 1 (computation)**.** Given a factory A and a product $x \in \{B, R\}^*$ as its input, the *computation* of A on x is defined as a sequence of configurations $(1, 1, x) =: c_0, c_1, \ldots$ where $c_k \implies c_{k+1}$ for all k.

We say A *accepts* x if there is a computation with $c_k = (m, m, y)$ for some $y \in \Sigma^*$ and $k > 0$. It *rejects* x if there is a computation with $c_k = (i, j, y)$ and $A_{ij} = \square$ for some $(i, j) \neq (m, m)$, $y \in \Sigma^*$ and $k > 0$.

Remark. We designate the $(1, 1)$-cell as the starting point of the computation, and (m, m)-cell the "accepting point". Clearly, any facility placed on that spot shall not be effective. This mildly different setting from the original game is obviously irrelevant for our purpose.

We also remind the reader of the possibility that our factory never halts – neither accepts nor rejects in finite number of steps.

As is the case for any computational model, we can charge one unit of time for each step of yielding relation. The total amount of time before halting is called the running time of $A(x)$. Similarly, the total amount of tape cells used during the computation is called the space consumption of $A(x)$. Then we can define the rather standard time and space complexity for our factory model.

2 The Computing Power of Factory Model

2.1 Equivalence to Turing Machines

Some levels in *Manufactoria* hints at the computing power of the factory model. Apart from typical regular languages such as $\{x \mid x \text{ has alternating reds and blues}\}$ and context-free languages such as $\{R^n B^n \mid n \in \mathbb{N}\}$, the model is in fact capable of dealing with recursive languages such as binary addition. We note that the factory model has a close affinity with single-tape Turing machines: the spatial placement of facilities in a factory correspond to state arrangement of a Turing machine. Also, that the readers read at the front and printers write at the back (i.e. in a *queue-fashion*) can be overcome by proper design. It thus comes at no surprise that the computing power of factory model coincides with Turing machines.

Theorem 1. The factory model is Turing complete. In fact, factories and Turing machines can simulate each other within a quadratic overhead in time.

Proof. It is obvious that there is some Turing machine M simulating A; otherwise *Manufactoria* would not run on our computers. The simulation can be done in quadratic time and linear space. The tape content of M is identical to that of A's, *preceded by* the the matrix A, a scratch area of length $O(\log m)$, and the coordinate pair (i, j) of the current configuration. In each step we read the first dot, look into the matrix A (indexed by two accumulators in the scratch area), and modifies the tape content according to A_{ij}. Each step costs at most linear time (in case A_{ij} contains a printer), so the overall complexity is as claimed.

Conversely, we shall design an $m \times m$ factory A that simulates a Turing machine M. Let $\delta : Q \times \{R, B, \sqcup\} \to Q \times \{R, B, \sqcup\} \times \{\leftarrow, \rightarrow\}$ be the transition function of M, where $Q = \{1, 2, \ldots, |Q|\}$ is the set of states. Suppose the starting state is 1 and, without loss of generality, *the* accepting state is $|Q|$.

We separate the $m \times m$ grid into $|Q|$ disjoint nonempty zones, $Z_1, Z_2, \ldots, Z_{|Q|}$. We require that $(1, 1) \in Z_1$, $(m, m) \in Z_{|Q|}$, and that Z_r solely consists of $\square$'s for any rejecting state r. We shall design A so that the product is sent to zone Z_q whenever M enters state q. Conceptually this is simple: we may place a reader ◆ at Z_q to branch according to the tape content. The red side is directed via ▲ 's to the state specified by $\delta(q, R)$, and similar for the blue and black sides.

However, the head of Turing machine M might hang somewhere in middle of the tape, while the reader and printer in factory A always sit at the front and back. The tape position pointed by A and M are usually inconsistent, so we must find a way to navigate through the tape in A. To accomplish this, A inserts a mark Y to the leftmost of the tape at startup; see Figure 1(a) for implementation. The mark always indicates the current head position of M. Besides, A appends another Y at the end of the tape to indicate boundary. As A proceeds to simulate M in state q:

1. It reads through the tape until a Y is met. During the process, it simply copies what it had read to the end of the tape, *with a lag of one symbol*. That is, the symbol l to the left of Y will not be appended to the end immediately; rather, this information is kept by branching. See Figure 1(b).

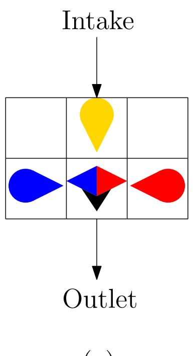
2. It removes the mark Y by a ◆ , and then branches by ◆ according to the next symbol a (which is exactly the one pointed by M.) Suppose $(q', a', d) := \delta(q, a)$.

   (1) If $d = \leftarrow$, then it prints Y, l and a' consecutively.

   (2) If $d = \rightarrow$, then it prints l, a' and Y consecutively.

3. Finally, it copies every remaining symbols until another Y is met. (This Y indicates the end of tape, and is copied to the end of tape as well.)

Now it remains to resolve the final twist: the possible crossing among the conveyor belts. Although this is allowed in *Manufactoria*, it is not modeled in our factory. Anyway, we shall show that crossings can be removed without affecting the computation. Here we utilise the vacant symbol G and the ability of ◆ . For a crossing site (i, j), we place on it a ◆ instead of ▲ . The black arrow is pointing to the same direction as the vertical flow, so it acts just like a ▲ in the vertical direction. As for the horizontal direction, we may



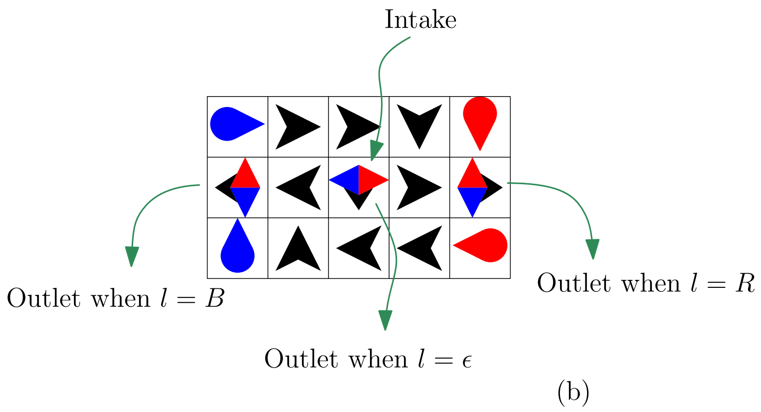


Figure 1: Two subroutines

stamp a G at the *front* of the tape before we send the product into the site. The reader will identify such a G and sends it horizontally.

The overall time complexity is quadratic since each step of simulation traverses the entire tape, and each crossing resolution requires an additional traversal. (Note that the number of crossings is a contant only related to m.) □

Corollary 2. The class P in both models coincide.

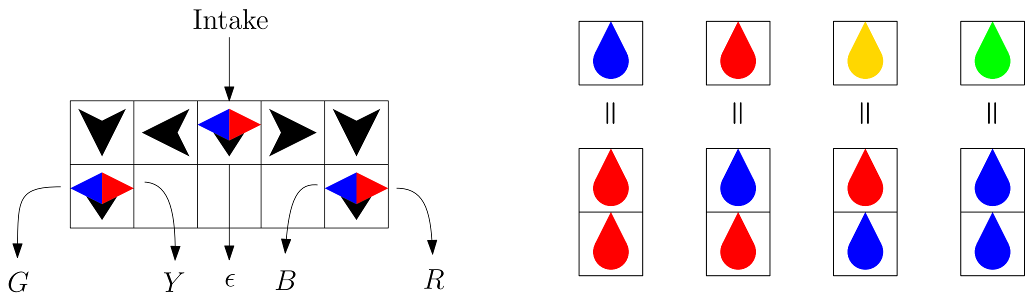
The simulation itself is interesting in some practical sense, for the factory model is extremely close to physical implementation of a pipelined factory. We list here several considerable advantages over a typical machine model:

- Standardising physical placement: the factory model can be directly embedded in a planar square grid.
- Enabling parallel computation: The factory model naturally allows pipelined processing by feeding a stream of products into the $(1, 1)$-cell, which is not achievable in a machine-based model. Moreover, the processing can be totally out of order without any harm to the correctness.

2.2 P versus PSPACE in Factory Model

It is clear from Theorem 1 that a Turing machine can simulate a factory within polynomial (in fact, linear) space overhead, but the converse is troublesome. Recall that our factory A must copy the entire tape string at each step of simulation, so it consumes a total space of $T(n) \cdot S(n)$ where T and S are time and space function of M. But for Turing machines, T can be in general an exponential function of S, which charges A an *exponential space overhead*. It is thus unclear from our construction that PSPACE in both models would coincide.

We shall pose strong evidence that the PSPACE in the two models are distinct. To simplify our discussion, we slightly restrict our model by requiring that all readers must



(a) (b)

Figure 2: Replacing readers and printers

consume a symbol each time they operate on a non-empty tape. For example, when a ◆ reads a Y, it *removes the* Y from the tape and branches to the black arrow.

Lemma 3. The restriction doesn't change the power and efficiency of the factory model.

Proof. We shall describe a method of simulating a general factory A by its restricted counterpart A'. The basic idea is to compress $\Sigma = \{B, R, Y, G\}$ into a binary alphabet. In specific, we adopt the following mapping: $R \mapsto RR$, $B \mapsto RB$, $Y \mapsto BR$ and $G \mapsto BB$.

At the very beginning, A' follows the subroutine in Figure 1(a) with a modification: it prints two symbols RB when it branches to the red side, and RR when it branches to the blue side. Finally, when the product comes out from the black side, the Y mark is automatically removed. From now on, we will ensure that the tape contains B's and R's only.

A' will have the same structure as A, but with all readers and printers substituted. Any reader is replaced by three ◆'s connected as shown in Figure 2(a). Any printer is split into two as shown in Figure 2(b). Obviously such construction only incurs a constant multiplier on the time and space complexity. $\square$

Theorem 4. $\mathsf{TIME}(f(n)) = \mathsf{SPACE}(f(n))$ in the factory model.

Proof. It is trivial that $\mathsf{TIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$, so it remains to show that $\mathsf{SPACE}(f(n)) \subseteq \mathsf{TIME}(f(n))$. Suppose for the sake of contradiction that there is a language $L \in \mathsf{SPACE}(f(n)) \setminus \mathsf{TIME}(f(n))$. Let A be *any* factory that decides L in space $f(n)$, so at most $f(n)$ steps are spent on printers. By our assumption, A does *not* run in time $f(n)$ for sufficiently long input x ($n := |x|$). We take x so that $A(x)$ runs for more than $T := m^2 \cdot (f(n) + 2))$ time. That is, $A(x)$ spares most of its time on conveyor belts or readers. In particular, x passes some conveyor or reader for more than $T/m^2 = (f(n) + 2)$ times. Let us denote that specific conveyor/reader position as θ. Now we distinguish two cases:

- **θ is a reader.** Since there are at most $f(n)$ symbols ever printed on the tape, and that each reading consumes a symbol, θ would read an empty tape for at least twice. But whenever it reads an empty tape, it always moves the product towards the black arrow and leaves the tape empty. That is, the configuration after this step is always

(i_0, j_0, ϵ) for some fixed (i_0, j_0). Clearly, θ sits in an infinite loop and A shall never halt, leading to a contraction.

- **θ is a conveyor.** Starting from θ, we follow the unique sequence of conveyors and printers until we are blocked by a reader. Formally, we define $(i, j) \vdash (i', j')$ if for $(f, o, s) := A(i, j)$, $f \notin \{$◆, ◆, $\square\}$ and $(i'j')$ is adjacent to (i, j) in direction o. We take the closure of (i, j) under $\vdash$, which gives us either a one-way path, or a path *plus* a cycle. If there is a cycle, then A clearly sits on an infinite loop. If there is no cycle, then the end of the path is a reader and the previous case applies. Either possibility leads to a contradiction.

$\square$

Corollary 5. $\mathsf{P} = \mathsf{PSPACE}$ in the factory model.