

Yale University
Department of Computer Science

P.O. Box 208205
New Haven, CT 06520-8285

THINK-A-DOT¹

Michael J. Fischer² Albert R. Meyer³

Michael S. Paterson⁴

YALEU/DCS/TR-1287

April 26, 2004

¹These notes originally appeared on September 18, 1973 in the form of notes for M.I.T. course 6.913, *Introduction to Theory of Computation*, co-taught by M. Fischer and A. Meyer. They were reissued on February 6, 1974, for M.I.T. course 6.045, *Introduction to Theory of Computation*, taught by A. Meyer. They were based in part on material from M.I.T. course 18.420, *Theory of Computation*, taught by M. Paterson in 1971 and by M. Fischer in 1972. We are grateful to Robert Moeser for transcribing the original typed notes to machine-readable form.

²Current affiliation: Yale University.

³Current affiliation: Massachusetts Institute of Technology.

⁴Current affiliation: University of Warwick.

THINK-A-DOT*

Michael J. Fischer[†]

Albert R. Meyer[‡]

Michael S. Paterson[§]

1 Introduction

The mini-theory of the Think-a-Dot game which we develop in this section will serve as a paradigm of the kinds of questions we will investigate and the style of mathematics we will use in our studies of abstract machines.

The Think-a-Dot itself is a plastic box into which marbles may be dropped. It has three holes along the top where the marble may enter and a hole on either side near the bottom from which the marble may exit. On the front are eight circular windows through which are visible yellow or blue flags. As a marble passes through the box, certain of the flags change from yellow to blue or from blue to yellow. (See Fig. 1.)

A little experimentation with the box shows that it has memory, for when a succession of balls are dropped through the middle hole, they exit two on the left, then two on the right, two on the left, two on the right, etc.

More experimentation would show that the pattern of colors visible through the windows and the hole into which the marble is dropped determines which hole the ball will exit from and what the new pattern of colors will be.

By taking the box apart, the action of the machine become clear. Inside are eight plastic flip-flops which may rest in one of two positions and to which are attached the two-color flags, so the color visible through the window tells the position of the flip-flop. When a ball hits a flip-flop in the yellow position or “state” it will go toward the left and change the flip-flop to the blue position. When it hits a blue flip-flop, it will go towards the right and change the flip-flop back to yellow. (See Fig. 2.) We will denote a flip-flop in the left state by “ L ” or  and similarly, we will denote the right state by “ R ” or .

[Note: The Think-a-Dot toy as it comes from the factory has the colors reversed from the above description on half of the flip-flops, making it more confusing to try to predict the outcome of dropping a ball. The machines used in class for demonstration have been modified so that all the flip-flops are the same. All further references to Think-a-Dot refer to the demonstration machine.]

*These notes originally appeared on September 18, 1973 in the form of notes for M.I.T. course 6.913, *Introduction to Theory of Computation*, co-taught by M. Fischer and A. Meyer. They were reissued on February 6, 1974, for M.I.T. course 6.045, *Introduction to Theory of Computation*, taught by A. Meyer. They were based in part on material from M.I.T. course 18.420, *Theory of Computation*, taught by M. Paterson in 1971 and by M. Fischer in 1972. We are grateful to Robert Moeser for transcribing the original typed notes to machine-readable form.

[†]Current affiliation: Yale University.

[‡]Current affiliation: Massachusetts Institute of Technology.

[§]Current affiliation: University of Warwick.

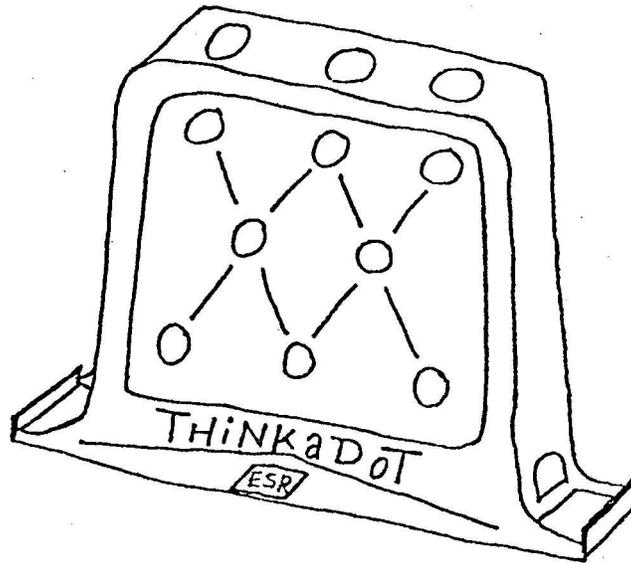


Figure 1: Think-a-Dot game.

We number the flip-flops from left to right beginning with the top row, and similarly going from left to right we label the input holes A , B and C and the output holes P and Q . The channels connect the flip-flops together as shown in Fig. 3.

The action of the machine is described quite simply: When a ball enters a flip-flop in state L , the state of that flip-flop changes to R and the ball exits through the left channel. Similarly, if the flip-flop was initially in state R , it changes to L and the ball exits through the right channel. It then travels through that channel until it hits the next flip-flop or comes out the bottom.

For example, if flip-flop 1 is in state L and a ball is dropped in hole A , the ball changes 1 to state R and drops straight through to flip-flop 6, bypassing both flip-flops in the second row. It is possible for a ball dropped in hole A to come out through hole Q , but only if flip-flop 1, 4 and 7 are all in state R .

At this point, we seem to have a complete understanding of the machine. We know the internal structure and we can predict its behavior knowing the initial state of all of the flip-flops and the sequence of marble drops. What more is there to know about the machine?

Quite a bit, as it turns out! There are numerous questions one may wish to ask about the machine for which the answers are not at all obvious, even knowing exactly how the machine works:

1. Starting from the all-yellow pattern, can one drop in marbles so as to make it all blue? Blue on top and bottom row and yellow in the middle? Alternating yellow and blue? If so, can one get back to the all-yellow pattern?
2. How many of the $2^8 = 256$ possible patterns can one get to starting from the initial state (all-yellow)?
3. Given two states s_1 and s_2 , what is the fewest number of marbles needed to get from s_1 to s_2 , assuming it is possible at all? Call this number the directed distance from s_1 to s_2 .
4. Is the distance from s_1 to s_2 always the same as the distance from s_2 to s_1 ?

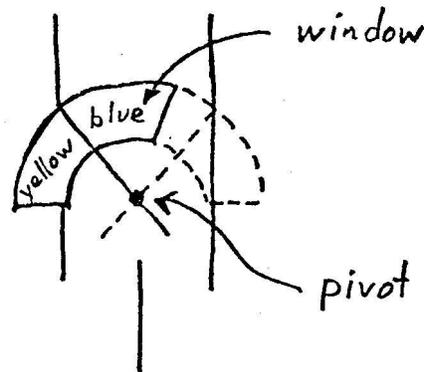


Figure 2: Flip-flop.

5. What is the largest distance between any pair of states for which the distance is defined?
6. If I turn the machine around so that you can't see the flip-flops and then shake it up to set the flip-flops in random positions, is it possible for you, by performing a number of experiments on the machine, to determine the state that the machine was in after I shook it up? If not, how close can you come? An *experiment* consists in dropping a ball in one of the three holes and noting which side it comes out.
7. If (6) is possible, how many experiments might be necessary?

This list is by no means exhaustive and it contains questions of varying difficulty. The purpose of a theory is to give one enough insight into the properties of the object of interest to enable such questions to be easily answered. We will proceed to answer the first five questions and in the process will develop such a theory.

2 State Accessibility

We begin with the problem of how to get from one state to another. Our first algorithm is based on the observation that a reverse diagonal, for example 3–5–7, acts like a binary counter. Each flip-flop represents one binary digit and the low-order digit is represented by the highest flip-flop (3 in the example). Zero is represented by state *R* and one by state *L*. Note that a ball taking the left exit from a flip-flop hits the next digit of the counter and may be thought of as a carry propagation, while a ball which takes the right exit from a flip-flop never again hits the same reverse diagonal.

Thus, successive balls through hole *C* cause the reverse diagonal 3–5–7 to cycle through each of the eight possible patterns of those three flip-flops. Similarly, successive balls through hole *B* cause the reverse diagonal 2–4–6 to cycle through all eight patterns, and balls through hole *A* cycle flip-flop 1 through its two possible patterns. (See Fig. 4.)

It is also clear that balls through holes *B* and *C* do not affect flip-flop 1, and balls through hole *C* do not affect the *B*-diagonal, 2–4–6. We thus have:

Algorithm 2–1 Assume the Think-a-Dot is in state *s*. To change it to state *s'*:

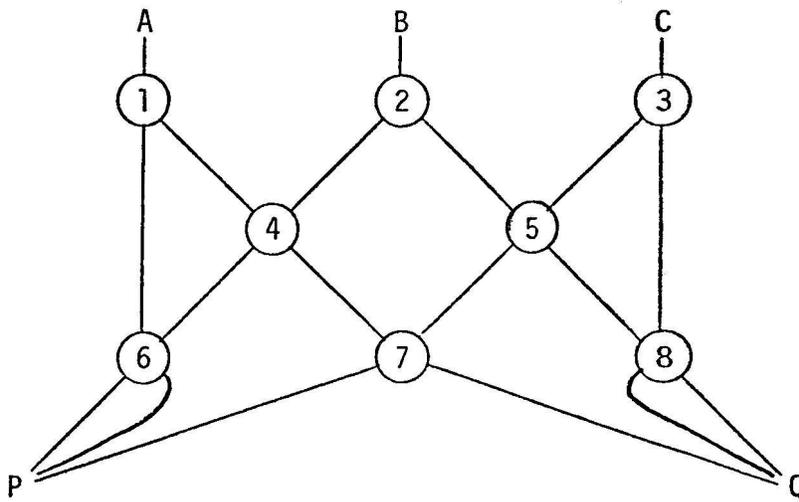


Figure 3: Channels between flip-flops.

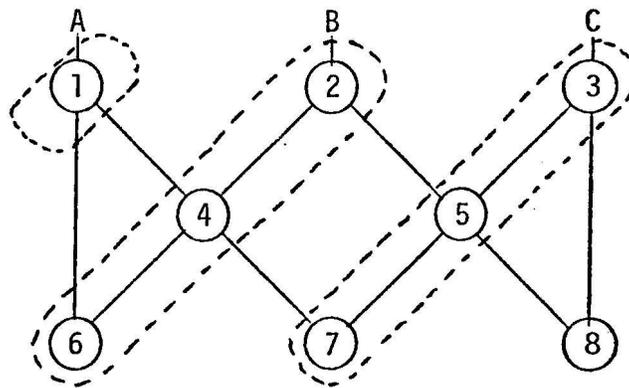


Figure 4:

1. Drop 0 or 1 balls through hole *A* to set flip-flop 1 to agree with s' .
2. Drop between 0 and 7 balls as needed through hole *B* to cause flip-flops 2, 4 and 6 to agree with s' .
3. Drop between 0 and 7 balls as needed through hole *C* to cause flip-flops 3, 5 and 7 to agree with s' .
4. If flip-flop 8 agrees with s' , succeed; otherwise fail.

Algorithm 2–1, when it works, may require the dropping of as many as 15 balls. [Exercise: Find two states s and s' such that Algorithm 2–1 uses that many balls.]

A better algorithm is given below:

Algorithm 2–2 Assume the Think-a-Dot is in state s . To change it to state s' :

1. Set the top row to agree with s' by dropping 0 or 1 balls as needed through each of holes A , B and C .
2. Set the middle row to agree with s' by dropping 0 or 2 balls as needed through each of holes A and C .
3. Set flip-flops 6 and 7 to agree with s' as follows:
 - (a) If neither agree, drop 4 balls through hole A .
 - (b) If 6 does not agree but 7 does, drop 4 balls through hole B .
 - (c) If 6 agrees but 7 does not, drop 4 balls through hole C .
4. If flip-flop 8 now agrees with s' , succeed; otherwise fail.

To see that Algorithm 2–2 works, one need only verify that the inputs accomplish the desired task and that no step affects the rows set by the previous steps. For example, 4 balls through hole A will result in 3 balls hitting flip-flop 6 and 1 ball hitting flip-flop 7, and hence both end up in the opposite position from which they started. (See Fig. 5.)

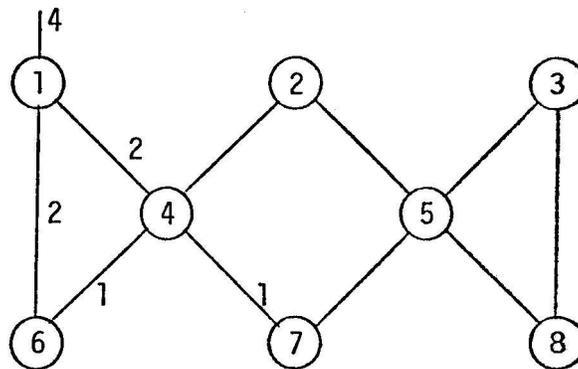


Figure 5: Four balls through hole A .

Algorithm 2–2, when it works, takes at most 11 balls, but again it sometimes fails.

An improvement of Algorithm 2–2 is to change step 2 to use 2 balls through hole B instead of 2 through A and 2 through C in the case where the latter would be necessary (that is, when both flip-flops 4 and 5 were wrong). This reduces the maximum cost to 9.

Still, these algorithms all have in common that they may fail. One could continue looking for new methods in hopes of finding one that would always work, but such is doomed to failure.

Theorem 2–3 *At most half of the states are reachable from any given state.*

Proof For a given state s , let $\text{par}(s)$ be the parity of the number of flip-flops in the top and bottom rows that are in state L , that is, $\text{par}(s) = 1$ if the number is odd and $\text{par}(s) = 0$ if the number is even.

No matter how the flip-flops are set and no matter what input hole is used, any ball causes exactly one flip-flop in the top row and exactly one in the bottom row to change state. Thus, the

total number in the top and bottom rows in state L changes by -2 , 0 or $+2$, and in each case, the parity remains the same.

By a simple induction, any state s' reachable from a state s has the same parity as s . But half the states have even parity while the other half have odd parity (Why?), so the half whose parity is opposite that of s cannot be reached from s . ■

Corollary 2-4 *There are exactly 128 states reachable from any given state.*

Proof Algorithms 2-1 and 2-2 each show that, starting from any state, we may set flip-flops 1 through 7 arbitrarily and hence there are at least $2^7 = 128$ different accessible states. From Theorem 2-3, at most that number are accessible. ■

Corollary 2-5 *Algorithms 2-1 and 2-2 will succeed in achieving the desired state if it is accessible at all from the current state.*

Corollary 2-6 *If it is possible to get from state s to state s' , then it is possible to get back from s' to s .*

Proof Both must have the same parity. ■

3 Equivalences of Input Sequences

We now develop the theory that will enable us to determine the minimum distance from a state s to a state s' and to find a sequence of inputs (i.e., ball drops) of that length taking s to s' .

Notation: We represent a sequence of ball drops by writing in order the names of the input holes, so $ABCB$ represents the dropping of a ball in a hole A , then one in B , then one in C , then one in hole B . The effect of a sequence X on a state s is to take the machine to a new state s' , and we write this by $sX = s'$. Thus, if $X = ACB$, $sA = s_1$, $s_1C = s_2$ and $s_2B = s_3$, then

$$\begin{aligned} sACB &= ((sA)C)B \\ &= (s_1C)B = s_2B = s_3. \end{aligned}$$

Definition 3-1 Two input sequences X and Y are *equivalent*, written $X \equiv Y$, if $sX = sY$ for every state s .

This definition may seem somewhat arbitrary at this point, but it is certainly not too strong for our purposes. Recall that we are trying to find the shortest sequence X such that $sX = s'$. Any sequence Y equivalent to X by the above definition will surely also take s to s' , and hence the sequence X we are seeking will be of minimal length in its equivalence class. [The equivalence class of X is the set of all sequences Y equivalent to X .]

A remarkable property of Think-a-Dot is that all sequences that take a given state s to a given state s' are equivalent, and thus the problem of finding the shortest sequence X such that $sX = s'$ is the same as the problem of finding the shortest sequence X equivalent to a given sequence Y (given that we already know by the previous section how to find *some* sequence Y taking s to s' when one exists).

Just as remarkable is the lemma that we will use to prove the above:

Lemma 3-2 *Think-a-Dot is commutative, that is, $XY = YX$ for all sequences X and Y .*

Proof We show that it is true when X and Y are single inputs. It follows for general X and Y by induction. [Exercise.]

Let X and Y be single inputs, s a state, and consider the paths P_X and P_Y through the machine taken by ball X and ball Y when dropped in the order first X , then Y . If the paths are totally disjoint (i.e., do not touch), then clearly the order in which the balls are dropped makes no difference, for the flip-flops altered by the drop of ball X do not affect the path taken by ball Y and vice-versa.

On the other hand, suppose the paths do intersect. Let J be the highest flip-flop contained in both paths. The parts of the paths P_X and P_Y above J are disjoint, so the order of the balls is immaterial up to flip-flop J . However, the effect of either XY or YX from point J on down is the same, for in both cases, the net result is the same as starting two balls at flip-flop J . (See Fig. 6.) ■

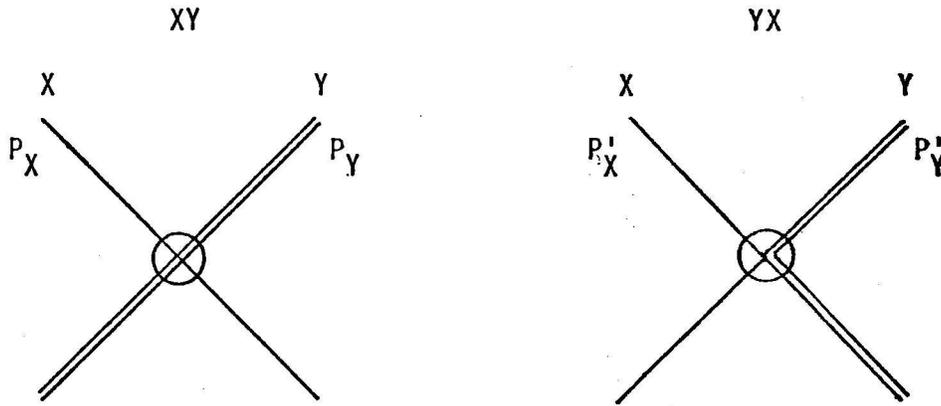


Figure 6: Intersecting paths.

Theorem 3-3 *Let $sX = sY$. Then $s'X = s'Y$ for all states s' accessible from s . [Recall: s' is accessible from s if there is a sequence Z for which $s' = sZ$.]*

Proof Let Z be a sequence taking s to s' , and let $s'_X = s'X$, $s'_Y = s'Y$, $s'' = sX = sY$, and $s''_Z = s''Z$. We may diagram these definitions as shown in Fig. 7. By Lemma 3-2, $XZ \equiv ZX$, and hence $s''_Z = sXZ = sZX = s'_X$. Similarly, $YZ \equiv ZY$, so $s''_Z = sYZ = sZY = s'_Y$. Hence, $s'X = s'_X = s''_Z = s'_Y = s'Y$ as was to be proved. ■

This theorem is somewhat less than satisfactory, for in order to conclude that X is equivalent to Y , one must show $s'X = s'Y$ for all states s' , not just for those accessible from s . We prove the stronger theorem by generalizing the machine, showing that it is true of the generalized machine, and then concluding that it must also be true of the original machine.

We generalize the machine by adding new inputs I_1, I_2, \dots, I_8 . The effect of input I_k on state s is the same as starting a ball at flip-flop k and then letting it fall through the machine, so $I_1 \equiv A$, $I_2 \equiv B$ and $I_3 \equiv C$. Using the additional inputs, it is easy to see that any state is accessible from any other. Moreover, Lemma 3-2 remains true of the new machine, and hence the proof of Theorem 3-3 still goes through, yielding:

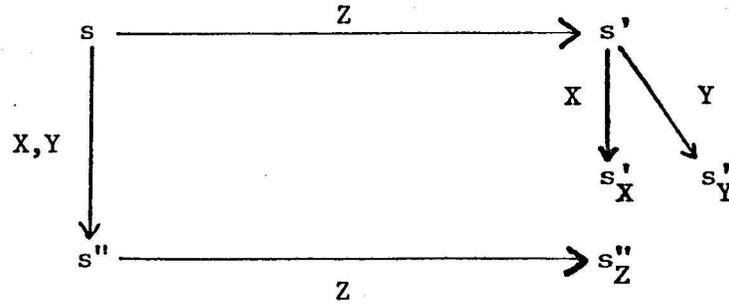


Figure 7: Commutative diagram.

Theorem 3–4 Let X and Y be any sequences of the inputs A, B, C and I_1, \dots, I_8 , and suppose $sX = sY$ for some state s . Then $X \equiv Y$.

Corollary 3–5 Let X and Y be sequences of the inputs A, B and C , and $sX = sY$ for some state s . Then $X \equiv Y$.

Corollary 3–5 says in effect that once you know what a sequence does to one state, then you know (i.e., can predict without further information about the sequence) what it does to any state. Thus, the equivalence class of a sequence is determined by its action on a single state.

Corollary 3–6 There are exactly 128 equivalence classes of input sequences.

Proof By Corollary 2–4, there are 128 states accessible from a given state s , so there must be at least 128 equivalence classes. By the above remarks, there are exactly that number. ■

As a minor digression, we now remark that one can now conclude the existence of unique inverses. We say that Y is an inverse of X if $sXY = sYX = s$ for all states s , or equivalently, if $XY \equiv YX \equiv I$, where I is the sequence of no inputs.

Let s be any state, and let $s' = sX$. By the parity argument in the proof of Theorem 2–3, s and s' have the same parity, and by Corollary 2–5, Algorithm 2–1 will succeed in finding a sequence Y taking s' to s , so $sXY = s$. Since $sI = s$, where I is the empty sequence of inputs, Corollary 3–5 shows that $XY \equiv I$. By Lemma 3–2, $YX \equiv XY \equiv I$, and hence Y is the inverse of X .

4 Identities and Minimal Length Canonical Forms

In this section, we show how to find a minimal length sequence Y equivalent to a given sequence X . This together with Algorithm 2–1 and Corollary 3–5 gives us a method for finding a shortest sequence taking s to s' , viz. find any sequence X taking s to s' and then find a minimal length sequence Y equivalent to X . Any other sequence Z taking s to s' must, by the corollary, be equivalent to X and Y and hence no shorter than Y .

Our basic tool will be the use of identities to transform a sequence X into an equivalent one Y of minimal length. That the transformations preserve equivalence is shown by the following:

Theorem 4–1 Let $P \equiv Q$. Then $UPV \equiv UQV$ for any sequences U and V .

Proof Let s be any state, and let $s' = sU$. Since $P \equiv Q$, then $s'P = s'Q$. Hence $sUPV = s'PV = (s'P)V = (s'Q)V = s'QV = sUQV$. This is true for any s , so $UPV \equiv UQV$. ■

We now give some identities true of Think-a-Dot.

Theorem 4-2 *The following identities hold.*

- (a) $AB \equiv BA, AC \equiv CA, BC \equiv CB$.
- (b) $A^8 \equiv B^8 \equiv C^8 \equiv I$. [A^8 means AAAAAAAAA, that is, a sequence of 8 A's.]
- (c) $A^2B^2C^2 \equiv I$.

Proof (a) follows from Lemma 3-2. To prove (b) and (c), we use the fact that if an even number of balls pass through a given flip-flop, half of them go each way from the flip-flop, and the flip-flop is left in the state that it started in. Thus, we need only prove that an even number of balls pass through each flip-flop.

We show in Fig. 8 that $A^8 \equiv I$. The other identities are proved similarly. Thus, when 8 balls

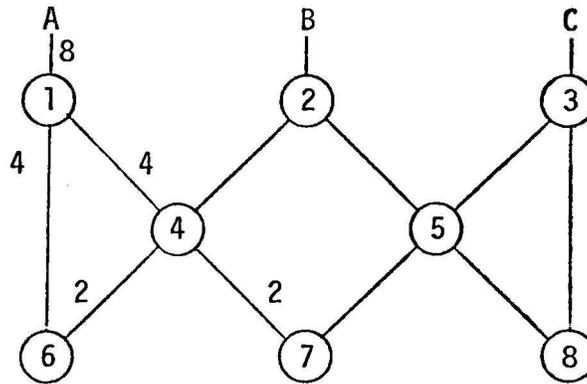


Figure 8: Proof of identity $A^8 = I$.

are dropped through hole A, 8 pass through flip-flop #1, 4 through #4, 6 through #6, 2 through #7, and none through #2, #3, #5 or #8. We see that an even number passes through each flip-flop, so the total state remains unchanged. ■

There are many other identities that one can write down, for example, $A^6 \equiv B^2C^2$. This of course can be obtained from (b) and (c) by two uses of Theorem 4-1 as follows:

$$A^6 \equiv A^6A^2B^2C^2 = A^8B^2C^2 \equiv B^2C^2.$$

One may ask whether or not every such identity true of Think-a-Dot can be derived from (a), (b) and (c). The next theorem answers that question in the affirmative, so we may speak of (a)–(c) as a *complete set of identities*.

Theorem 4-3 *The identities of Theorem 4-2 are a complete set of identities for Think-a-Dot.*

Proof The identities are sufficient to show any sequence X equivalent to a sequence of the form $A^i B^j C^k$ where $0 \leq i \leq 1, 0 \leq j, k \leq 7$. To do this, first use (a) to put X into the form $A^i B^j C^k$. Then repeatedly use the identity $A^2 \equiv B^6 C^6$ (derivable from (b) and (c)) followed by applications of (a) to get a sequence $A^{i'} B^{j'} C^{k'}$ where $0 \leq i' \leq 1$. Now repeated applications of (b) result in the desired form.

There are only 128 sequences $A^i B^j C^k$ with $0 \leq i \leq 1$ and $0 \leq j, k \leq 7$, and we have just shown that any sequence X can be shown equivalent to one of them by use of the identities (a)–(c). By Corollary 3–6, there are 128 distinct equivalence classes, so no pair of sequences of this form are equivalent. Hence, if $X \equiv Y$, then X and Y are both equivalent to the *same* sequence of the form $A^i B^j C^k$, and those two equivalences can be shown by the identities (a)–(c). Together they show $X \equiv Y$. ■

Finally we show how to apply the identities (a)–(c) to get a sequence of minimal length.

Definition 4–4 Let X be a sequence of exactly i occurrences of A , j occurrences of B , and k occurrences of C . We call X a *1–3–5 sequence* if:

- (i) $\min(i, j, k) \leq 1$;
- (ii) $\text{med}(i, j, k) \leq 3$;
- and (iii) $\max(i, j, k) \leq 5$.

[Min, med, and max of (i, j, k) are defined by arranging i, j and k in increasing order. Then $\min(i, j, k)$ is equal to the first one in that order, $\text{med}(i, j, k)$ is equal to the second one, and $\max(i, j, k)$ is equal to the third one. For example, $\min(2, 3, 2) = \text{med}(2, 3, 2) = 2$ and $\max(2, 3, 2) = 3$.]

Theorem 4–5 Consider the three sets of identities:

- (I) $A^2 B^2 C^2 \equiv I$;
- (II) $A^4 B^4 \equiv C^4$; $A^4 C^4 \equiv B^4$; $B^4 C^4 \equiv A^4$;
- (III) $A^6 \equiv B^2 C^2$; $B^6 \equiv A^2 C^2$; $C^6 \equiv A^2 B^2$.

One of these identities can be applied to X in the left-to-right order (i.e., by substituting the right side for the left in X) together with the use of commutativity if and only if X is not a 1–3–5 sequence.

Proof (\Rightarrow). Suppose one of these identities can be applied to a string X in the left-to-right direction. If (I) can be applied, then X violates condition (i) of the definition of a 1–3–5 sequence. Similarly, if (II) or (III) can be applied, then X violates condition (ii) or (iii) respectively. In each case, X is not a 1–3–5 sequence.

(\Leftarrow). Conversely, if X is not a 1–3–5 sequence, then one of the conditions (i), (ii) or (iii) must fail, in which case the corresponding identity can be applied. ■

Corollary 4–6 Any method of repeatedly applying the identities (I)–(III) in the left-to-right direction to X must eventually terminate in a 1–3–5 sequence.

Proof Each application reduces the length of the sequence. By Theorem 4–5, another application is always possible until the sequence becomes a 1–3–5 sequence. ■

We now show the 1–3–5 sequences are precisely the ones we are interested in.

Theorem 4–7 *A sequence X is minimal iff it is a 1–3–5 sequence.*

Proof Suppose X is minimal. Then none of the identities (I)–(III) is applicable, since each of them shortens the string to which it is applied and X is already minimal. Hence, by Theorem 4–5, X is a 1–3–5 sequence.

Conversely, suppose X is a 1–3–5 sequence, and suppose it is not minimal. Then there is some minimal sequence Y equivalent to X , and $Y \neq X$ since it is strictly shorter. By the forward direction of this theorem just proved, Y is a 1–3–5 sequence and hence X and Y are two distinct 1–3–5 sequences that are not equivalent.

The following lemma shows that the above leads to a contradiction, completing the proof. ■

Lemma 4–8 *No two distinct 1–3–5 sequences are equivalent.*

Proof By Corollary 4–6, every sequence is equivalent to a 1–3–5 sequence, so there is at least one 1–3–5 sequence in each equivalence class. By Corollary 3–6, there are exactly 128 equivalence classes. Thus, if we can show that there are at most 128 distinct 1–3–5 sequences, it will follow that there is exactly one per equivalence class and hence no two 1–3–5 sequences are equivalent.

It remains to count the number of 1–3–5 sequences. We do this by counting separately:

- (1) The number of sequences satisfying condition (iii).
- (2) The number of sequences satisfying (iii) but violating (i).
- (3) The number of sequences satisfying (i) and (iii) but violating (ii).

The number of 1–3–5 sequences is then equal to (1) – (2) – (3).

The number of sequences (I) is 6^3 , for i, j and k can each assume any of the six values between 0 and 5.

The number of sequences (II) is 4^3 , for to satisfy (iii) but violate (i), it must be that i, j and k are each in the range 2 through 5.

The number of sequences (III) is $3 \cdot 2^3$. A sequence satisfying (i) and (iii) but violating (ii) must have one of i, j or k equal to 0 or 1, while the other two numbers are both equal to 4 or 5. There are 3 choices as to which of i, j or k shall be ≤ 1 . Once that decision is made, there are two possible values for each of the three numbers.

Adding this all up, we get that the number of 1–3–5 sequences = $6^3 - 4^3 - 3 \cdot 2^3 = 216 - 64 - 24 = 128$. ■