USING COMPUTABLE NUMBERS TO SOLVE THE DECISION PROBLEM

KATHERINE STEINER

This paper gives an overview of Alan Turing's 1936 paper *On Computable Numbers, with an Application to the Entscheidungsproblem* [3], the paper in which Turing introduces his now famous hypothetical Turing machines. We'll first give some history of the problem Turing was trying to solve, and then give a detailed synopsis of his first major result: that the computable numbers are countably infinite. With a better understanding of Turing's machines, we'll then outline how Turing built on these ideas to solve a decision problem.

1. Introduction to the decision problem

In August 1900, David Hilbert gave a speech at the Second International Conference of Mathematics, and posed problems he considered mathematically interesting to encourage mathematicians to study them. His tenth problem, "Entscheidung de Lösbarkeit einer diophantischen Gleichung," in English reads:

> 10. Determination of the Solvability of a Diophantine Equation[1]
>
> Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: *To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.* [1]

We note that Hilbert isn't asking for a general process to *solve* Diophantine equations, but a process to determine whether or not they *can be* solved at all. In fact, in the statement of the problem, Hilbert already assumes that a general process exists. In Turing's paper, he proves a result, related to a more general decision problem, the Entscheidungsproblem:

Theorem 1. *The Entscheidungsproblem can have no solution, i.e. no general process for determining solvability of a statement given a set of logical axioms can exist.*

This problem is related to Gödel's Incompleteness Theorem:

Theorem 2 (Gödel's Incompleteness Theorem)**.** *Given a consistent set of axioms, there exist statements which cannot be proven to be true or false.*

[1]A Diophantine equation is a polynomial equation, usually with multiple indeterminants, with integer coefficients in which only integer solutions are allowed. For example, $x^3 + y^3 = z^3$ is a Diophantine equation with no integer solutions, by Fermat's Last Theorem.

From Gödel, we know unprovable statements exist, so the Entscheidungsproblem is to find a process to decide which are these statements. In the first part of Turing's paper, he develops a theory of computable numbers, which we will define carefully later in Definition 4, and proves the following result.

Theorem 3. *The computable numbers are countably infinite.*

In the second part of his paper, he shows that this theory can be used to prove Theorem 1. While certainly an interesting and substantial result, Turing's paper did not actually answer Hilbert's problem, but a related one. Turing's paper addresses the decision problem in the context of the provability of a logical statement, while Hilbert's problem is concerned with the solvability of an algebraic equation. After Turing's paper was published, mathematicians suspected that Hilbert's tenth problem would also have no solution, but it was not until 1970 that Hilbert's tenth problem was finally answered when Yuri Matiyasevich proved that no general process for determining the solvability of Diophantine equations can exist.

Now that we understand the problem, and the inspiration for the problem, that Turing was trying to solve, we can summarize the proof of his first major result, Theorem 3, which states that the computable numbers are enumerable. This proof will introduce us to the ideas of Turing's computing machines. Once we understand these theoretical machines, we can give a synopsis of the rest of Turing's paper.

2. Computable numbers

In Turing's paper, he shows that no general process exists for determining the solvability of a statement can exist. At face value, it seems like this should require some kind of very abstract argument. Instead, Turing begins his paper by describing hypothetical, but fairly simple, machines which can preform a limited set of operations. Later he'll use these machines to prove Theorem 1.

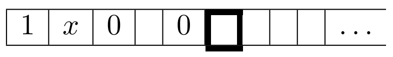
The following is a more detailed synopsis of Turing's proof that the computable numbers, which he defines as follows, are countable. In Turing's eleven section paper, this proof begins in Section 1 and finishes in Section 5.

Definition 4. *A* ***computable sequence*** *is a sequence which can be written down by a machine given a finite number of instructions. A computable sequence of 0's and 1's (e.g. 010101...) corresponds to the* ***computable number*** *written in binary obtained by prefacing the sequence with a binary point (0.010101... which in base ten is 1/3). The numbers between 0 and 1 which can be obtained in this way are computable numbers. Computable numbers also include numbers which differ by an integer value from the computable numbers between 0 and 1.*

Throughout Turing's paper, he mostly uses the language of about computable sequences, rather than computable numbers, to avoid confusion, and only talks about real numbers. It follows immediately from the definition that there are infinitely many computable numbers. (Informally, if we can accept that there is at least one computable number a between 0 and 1, then since the computable numbers also include numbers which differ by an integer value from a, and there are infinitely many integers, then there are infinitely many computable numbers.) To show that these are enumerable, Turing describes computing machines, which can each compute exactly one computable number, and then shows that we can assign a unique "description number" to each computing machine. Thus we can construct an injection to $\mathbb{Z}$, so the computable numbers are countably infinite.

Turing first describes a general machine as follows. The machine has an infinitely long tape, which is originally blank, divided into squares, which can each contain a symbol. At any moment, the machine "scans" the rth square, whose symbol (if there is one) he denotes by $\mathfrak{S}(r)$. At any moment, the machine has one of a finite number of ***m*-configurations**, which when Turing first introduces them, are denoted by Fraktur letters $\mathfrak{b}, \mathfrak{c}$, etc. An m-configuration describes what the machine can do at this time. The combination of a $\mathfrak{S}(r)$ and a m-configuration Turing refers to as simply a configuration. A machine can: erase its current scanned symbol, print a new symbol, change the scanned square by one to the right or left, or change its current m-configuration. Note that the machine has no concept of "memory" as do modern computers we are familiar with- the behavior of Turing's computing machines depend entirely on the machine's currently scanned symbol $\mathfrak{S}(r)$ and current m-configuration. Also, this kind of computing machine can only compute one sequence; it is not a reprogrammable computer.

For the purposes of this paper, Turing is interested in automatic computing machines: machines whose behavior is entirely determined by its current configuration (the machine always "knows what to do," i.e. its motion is well defined, and its next step is never ambiguous), and which print symbols "of the first kind" (which are 0's and 1's) and "of the second kind" (which will be used as rough notes to aid in computation, could be anything other than 0's or 1's, and in particular Turing uses x and ə in his examples). The squares of the tape alternate between E-squares and F-squares. Only symbols of the second kind can be printed on E-squares, and only E-squares may be erased. Only symbols of the first kind can be printed on F-squares, and the symbols on the F-squares form a continuous sequence of 0's and 1's which corresponds to a real number between 0 and 1, printed in binary. If the machine prints 01, then we say that the machine has computed the binary number 0.01, which in base 10 is 1/4. An example tape, at some point in a computing machine's operation, might look something like the following, where the currently scanned square is heavily outlined.



Finally, Turing introduces the idea of a **circle-free machine**, which is a machine which prints 0's and 1's forever. For example, he wants the machine computing the sequence of 0's and 1's corresponding to the binary representation of the decimal number 1/4 to print 01000 . . . , and continue printing 0's forever. If, at some point, the machine can no longer print any more symbols of the first kind, then we say that the machine is circular[2].

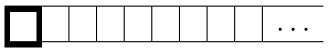
[2]See Appendix, example 1 for a machine which is circular.

The paper then gives examples of computing machines, one which computes the sequence 010101... which corresponds to the binary number 0.010101... and to the base 10 rational number 1/3, and one which computes the much more complicated sequence 010110111011110111110... (a 0, one 1, 0, two 1's, 0, three 1's, etc). He provides tables relating the current configuration of the machine (m-configuration and symbol pair) to what operations the machine takes (moving to the right or left, printing or erasing symbols) and to which m-configuration the machine switches. He also writes about skeleton tables, abbreviations of the tables of m-configurations for common processes, such as copying down sequences or comparing sequences. These are not essential to the proof, but do make later work easier.

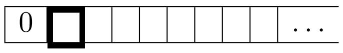
The following is the table of the (very simple) machine which prints the sequence 010101... which corresponds to 1/3, provided by Turing on page 233[3]. $P0$ means the machine prints a 0, $P1$ means that the machine prints a 1, and R means that the machine moves to the right. All of Turing's machines begin in the m-configuration $\mathfrak{b}$.

m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$

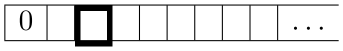
At the start, the tape is blank, and the currently scanned square is heavily outlined:



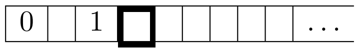
So, this machine starts with m-configuration $\mathfrak{b}$ and there is no scanned symbol. It therefore prints a 0, moves one square to the right, and switches to m-configuration $\mathfrak{c}$. The tape is now:



Now the machine has m-configuration $\mathfrak{c}$ and there is no scanned symbol. The machine therefore moves one square to the right (we're skipping a square because Turing wants his machines to only print on F-squares, and we're currently on an E-square) and switches to m-configuration $\mathfrak{e}$. The tape is now:



Now the machine has m-configuration $\mathfrak{e}$ and there is no scanned symbol. Therefore, the machine prints a 1, moves one square to the right, and switches to m-configuration $\mathfrak{f}$. The tape is now:



[3]On the very next page, Turing rewrites this table in a much more compact form, which is given in the Appendix, example 2, but leaving the table as given here makes our outline, and later work, easier to follow.

Now the machine has m-configuration $\mathfrak{f}$ and there is no scanned symbol. Therefore, the machine moves one square to the right and switches to m-configuration $\mathfrak{b}$. This process continues. By following along with the table, we can see that this machine does indeed print the sequence 010101... forever, and therefore this table, which has a finite number of instructions, defines the machine which computes the number 1/3. Thus, 1/3 is a computable number. Also, since this machine does print symbols of the first kind (0's and 1's) forever, this computing machine is circle-free.

This machine's table is very simple because the operations necessary to compute 1/3 according to Turing's requirements are straightforward and repetitive[4]. To compute other sequences, such as his 010110111011110111110... example, we need to use the E-squares for scratch work, print other figures, and erase figures. We'll now describe how to construct a general machine's **description number**, an integer which completely describes a machine, and also construct our 1/3 machine's description number. However, since our 1/3 machine is so simple, some of the steps that we describe will seem trivial.

By Turing's definition, a number is computable when its corresponding sequence can be printed by a computing machine. So any computable number can be described in terms of the table of m-configurations which define the machine that computes its sequence. Now Turing wants to standardize these tables, by rewriting these m-configurations. Where previously we had the machine erasing the symbol on the square, we'll say that the machine prints a blank, and where we previously had the machine not printing anything, we'll say that the machine prints the same symbol that is already on that square, or prints a blank if there was already no symbol on that square. So now we can reduce a line in the m-configuration table to one of three standard forms: the machine moves left and prints a symbol, the machine moves right and prints a symbol, or the machine does not move but prints a symbol. Rename the m-configurations so that we can easily enumerate them; let them now be $q_1, \dots, q_R$ where the first m-configuration the machine is in, $\mathfrak{b}$, becomes q_1. Also, enumerate the finite number of possible symbols we could print by letting S_0 be a blank, $S_1 = 0$, $S_2 = 1$, $S_3 = ə$, and so on.

The table for our 1/3 computing machine is now:

m-config.	*symbol*	*operations*	*final m-config.*
q_1	S_0	PS_1, R	q_2
q_2	S_0	PS_0, R	q_3
q_3	S_0	PS_2, R	q_4
q_4	S_0	PS_0, R	q_1

Now, for each line in the table, we can form an expression in the form $q_i S_j S_k L q_m$, which corresponds to the line which says that when the machine has m-configuration q_i and the scanned square is S_j, print S_k, move left (for a move to the right, this part of

[4]In particular, this machine never prints symbols other than 0's and 1's, the currently scanned symbol never affects which operations the machine carries out, and this machine never erases a symbol. See Appendix, example 3 for a machine for which this is not the case.

the expression is R, and for no move this part is N), and switch to m-configuration q_m. Take all such expressions of a computing machine and concatenate them, separated by semicolons; this is now a complete description of the machine.

For our 1/3 computing machine, we have

$$q_1 S_0 S_1 R q_2; q_2 S_0 S_0 R q_3; q_3 S_0 S_2 R q_4; q_4 S_0 S_0 R q_1; .$$

Now take this expression, and replace q_i with a D followed by A repeated i times and replace S_j with a D followed by a C repeated j times. Now we have a expression made up of the capital letters A, C, D, L, R, N, and semicolons. Turing calls this the machine's standard description.

For our 1/3 computing machine, we get:

$$DADDCRDAA; DAADDRDAAA; DAAADDCCRDAAAA; DAAAADDRDA; .$$

Finally, if we take this expression and replace A with 1, C with 2, D with 3, L with 4, R with 5, N with 6, and semicolons with 7, we obtain "a description of the machine in the form of an arabic numeral," which Turing calls the description number of the machine. Note that this number completely describes the behavior of the machine. He calls the machine whose description number is n, $\mathcal{M}(n)$, and an arbitrary computing machine just $\mathcal{M}$.

Thus the description number of our 1/3 computing machine is:

$$31332531173113353111731113322531111731111335317.$$

Observe that a particular description number describes exactly one machine, which computes a unique sequence corresponding to a unique computable number. However, we could easily define many computing machines which all compute the same sequence[5]. Thus, "to each computable sequence there corresponds at least one description number, while to no description number does there correspond more than one computable sequence. The computable sequences and numbers are therefore enumerable," Turing concludes on page 241.

3. A short discussion of the proof of Theorem 3

In Section 2, we gave an outline of Turing's proof of Theorem 3, that the computable numbers are enumerable. If the computable numbers are enumerable, then they are a strict subset of the real numbers. In Turing's paper's section 10, he names some classes of numbers which are computable: $\mathbb{Q}$, algebraic numbers over $\mathbb{Q}$, numbers which can be expressed as power series (e.g. π, a transcendental number), and zeros of Bessel functions. So what would an uncomputable number look like? In Section 8, Turing uses the idea of Cantor's diagonal proof that the real numbers are uncountably infinite to show that there can be no general process for determining whether an arbitrary computing machine is circle-free, and along the way he defines a sequence of 0's and 1's, which he calls β, which is uncomputable. He constructs this β from a list of other

[5]See Appendix, example 4 for a different computing machine which also computes 1/3.

computable sequences in the same way that Cantor constructs his real number from a list of real numbers. But while Cantor's new number must also be a real number that is not on the original list, a β constructed in this way he shows is not necessarily computable. Another uncomputable number is Chaitin's constant, which is related to the probability that an arbitrary machine is circle-free (although this is usually defined in the context of the halting problem- the question of whether or not an arbitrary program will run forever or eventually stop). Since this number is uncomputable, it's also one of the few numbers which has been proven to be transcendental, by Theorem 5.

Theorem 5. *All uncomputable numbers are transcendental (over $\mathbb{Q}$).*

Proof. This follows directly from Turing's claim that all algebraic numbers over $\mathbb{Q}$ are computable. If a number is not computable, then it cannot be algebraic, so it must be transcendental. Note, however, that not all computable numbers are algebraic. $\square$

4. Paper overview

Now that we have some understanding of Turing's computing machines, we can outline the rest of his proof. In Section 2, we went over how Turing defines computable numbers and describes computing machines. We showed how each computing machine computes one computable number, and that the computable numbers are countably infinite. Recall that Turing is interested in "circle-free" machines- machines which print the 0's and 1's forever. Then, Turing uses the concepts of computing machines to describe a universal machine, a special computing machine which can compute any computable number (these are now known as **Turing Universal Machines**, or just Turing Machines). He next proves that there is no general process for deciding whether or not a particular computing machine is circle-free. From this, he shows that there can be no general process for deciding whether or not an arbitrary computing machine ever prints a particular symbol, say 0.

Turing then links the concept of computing machines to logical problems. He claims that determining whether a given integer n has a particular property is equivalent to computing a sequence whose n-th figure is 1 if n has that property, and 0 if n does not. Up until this point in the proof, Turing hasn't yet proven that his definition of computable numbers includes all numbers which we naturally would like to be computable. He first provides an argument appealing to intuition, then a second argument which uses methods of Hilbert function calculus, and finally a third argument which combines elements of the first two. In particular, he claims the algebraic numbers over $\mathbb{Q}$, some transcendentals over $\mathbb{Q}$ like π and e, and the zeros of Bessel functions are computable.

Finally Turing uses these tools to answer the Entscheidungsproblem. He claims that for each computing machine, that we can use prepositional logic to construct some logical formula (i.e., a statement) based on the m-configurations of a computing machine. Turing describes how this formula is to be constructed, and shows that there is a general process for determining whether the formula of the machine is provable *if and only if* there is a general method for determining whether the computing machine ever prints a particular symbol. But he has already shown that there is no general process for deciding whether or not an arbitrary computing machine ever prints a particular symbol. Therefore, he concludes, the Entscheidungsproblem has no solution. There is no general process for determining the solvability of a statement given a set of logical axioms.

5. Legacy

Today, Turing is considered to be a key figure in the development of computer science. However, he wrote his paper on computability, developing the computing machines which would become known as universal Turing machines, for the mathematical community. In some ways his computing machines are far removed from today's computers since his have no concept of memory or storage, and he isn't concerned with the time it takes for his machines to carry out operations, for example. And yet, it can be shown that any computer that can emulate a Turing Machine is a "universal" computer, and any universal computer can emulate any other universal computer. Thus, in a way, modern computers and Turing's hypothetical computing machines are the same. In 1936, far in advance of the building of anything resembling a modern day computer, Turing had already proven that there are some things that no computer will ever be able to do.

References

[1] Hilbert, David. *Mathematical problems.* Bulletin of the American Mathematical Society. Vol 8 (1902), no. 10, 437-479.

[2] Petzold, Charles. *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine.* Wiley Publishing, Inc., 2008.

[3] Turing, A. M. *On Computable Numbers, with an Application to the Entscheidungsproblem.* Proceedings of the London Mathematical Society, Col s2-42, Issue 1, 1 January 1937, 230-265.

Appendix

To clarify some concepts, additional examples of Turing computing machines are provided in this section. These are chosen for the reader to compare and contrast against the machine which computes 1/3, used as an example in Section 2.

(1) An example of a machine which is circular is given by the following table. This machine is circular because it does not print 0's and 1's forever.

m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{c}$

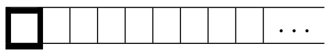
(2) We can rewrite the table for the 1/3 computing machine in a much more compact form, if we allow the machine to move more than one square at a time.

m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0$	$\mathfrak{b}$
$\mathfrak{b}$	0	$R, R, P1$	$\mathfrak{b}$
$\mathfrak{b}$	1	$R, R, P0$	$\mathfrak{b}$

So we only have one m-configuration, but the machine performs different operations based on the currently scanned square.

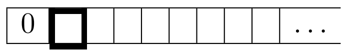
(3) Recall that the configuration of a machine, which is a combination of a m-configuration and the currently scanned symbol, define what operations (printing, moving, etc) the machine will take. The 1/3 computing machine that we used as an example in Section 2 never printed a symbol other than a 0, never erased a symbol, and always had the currently scanned square blank. The following machine is an example of a machine which will print a x, erase it, and scan a non-blank square.

As always, we begin with a blank tape, and in m-configuration $\mathfrak{b}$. $P0$ means the machine prints a 0, $P1$ means it prints a 1, Px means it prints a x, E means the machine erases the currently scanned square, and R means that the machine moves to the right. Here is our tape at the start, where the scanned square is heavily outlined:

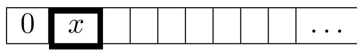


m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	Px	$\mathfrak{d}$
$\mathfrak{d}$	x	E, R	$\mathfrak{e}$
$\mathfrak{d}$	None	R	$\mathfrak{f}$
$\mathfrak{e}$	None	$P0, R$	$\mathfrak{d}$
$\mathfrak{f}$	None	$P1, R$	$\mathfrak{d}$

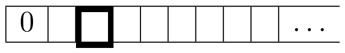
Since the machine is in m-configuration $\mathfrak{b}$ and there is no scanned symbol, then the machine prints a 0, moves to the right, and switches to m-configuration $\mathfrak{c}$. The tape is now:



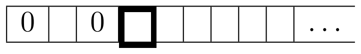
Now the machine is in m-configuration $\mathfrak{c}$ and there is no scanned symbol. Therefore, the machine prints a x (the current square is an E-square, so printing symbols other than 0's and 1's is okay) and switches to m-configuration $\mathfrak{d}$. The tape is now:



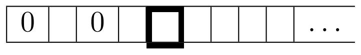
Now the machine is in m-configuration $\mathfrak{d}$, and the scanned symbol is x. Therefore, the machine erases the square (the current square is an E-square, so erasing is okay), moves to the right, and switches to m-configuration $\mathfrak{e}$. The tape is now:



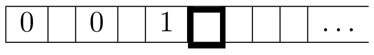
Now the machine is in m-configuration $\mathfrak{e}$ and there is no scanned symbol. Therefore, the machine prints a 0, moves to the right, and switches to m-configuration $\mathfrak{d}$. The tape is now:



Now the machine is in m-configuration $\mathfrak{d}$ and there is no scanned symbol. Recall that the machine has been in m-configuration $\mathfrak{d}$ before, but with a different scanned symbol (x). This time, the table tells us that the machine moves right and switches to m-configuration $\mathfrak{f}$. The tape is now:



Now the machine is in m-configuration $\mathfrak{f}$ and there is no scanned symbol. Therefore, the machine prints a 1, moves right, and switches to m-configuration $\mathfrak{d}$. The tape is now:



Now the machine is in m-configuration $\mathfrak{d}$ and there is no scanned symbol. Therefore, the machine moves right and switches to m-configuration $\mathfrak{f}$. By inspection, we see that now the machine will switch between m-configuration $\mathfrak{d}$ and $\mathfrak{f}$, and continue to print 1's on alternating squares. Thus, this machine computes the binary sequence 0011111 . . . .

The operations of this machine might be a little confusing. We could define another machine which computes exactly the same sequence, but without having to bother with printing and erasing a x. Perhaps you can try to write out the table of such a machine! But this table serves as a fairly short example of a machine which carries out different operations depending on the scanned symbol, and as an example of a machine which prints symbols other than 0's and 1's, and which erases symbols.

For a much more complicated example, see Turing's table for a machine which computes 010110111011110111110 . . . (0, one 1, 0, two 1's, 0, three 1's, etc) on page 234 of his paper, or example 5, below.

(4) An easy way to define another computing machine which also computes 1/3, but is different than the one we used as an example in Section 2, is to add an unnecessary line to the machine's table. For example:

m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$
$\mathfrak{g}$	None	L	$\mathfrak{b}$

This machine is exactly the same as the one we used as an example, except that it also has m-configuration $\mathfrak{g}$. But since no other m-configuration switches to it, clearly m-configuration $\mathfrak{g}$ will never be used. If we let $\mathfrak{g}$ be q_5 when we construct the machine's description number, we'll get the same description as before, but with $q_5S_0S_0Lq_1$; added at the end. The description number of this machine is then:

31332531173113353111731113322531111731111335317311111334317

which is the same as our previous example, with 311111334317 added on at the end.

(5) Here is one last example of a computing machine. The table has m-configurations which perform many more operations at a time than what is technically allowed, but we could simply define more m-configurations to put this table back in the form Turing required. But the idea of the machine is the same, so see if you can figure out what sequence it will print! The operations are defined as usual, and the machine begins in m-configuration q_1 with a blank tape.

m-config.	*symbol*	*operations*	*final m-config.*
q_1	None	$P1, R, R$	q_2
q_2	None	$P0, R, R$	q_3
q_3	None	$P1, R, Px, R$	q_4
q_4	None	$P1, R, R, P1, R, R, P1, R, R$	q_5
q_5	None	$P1, R, R, P0, R, R, P1, R$	q_6
q_6	None	L, L	q_6
q_6	x	E, R	q_7
q_6	y	E, R	q_8
q_7	None	$L, Py, R, P0, R, R, P0, R, R, P0, R, R$	q_5
q_7	0	R, R	q_7
q_7	1	R, R	q_7
q_8	None	$L, Px, R, P1, R, R, P1, R, R, P1, R, R$	q_5
q_8	0	R, R	q_8
q_8	1	R, R	q_8