بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ــ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

نظریهٔ زبان‌ها و ماشین‌ها

حسین فلسفین

Part 1: Variants of Turing Machines

Alternative definitions of TMs abound, including versions with multiple tapes or with nondeterminism. They are called variants of the Turing machine model. The original model and its reasonable variants all have the same power—they recognize the same class of languages. In this session, we describe some of these variants and the proofs of equivalence in power. We call this invariance to certain changes in the definition robustness. TMs have an astonishing degree of robustness.

Sipser's definition of a TM is a typical and standard one, but it is by no means the only one. We can think of some variants of that definition by somehow generalizing the definition. To show the equivalence, we simply show that we can simulate one variant by the other.

In this session, we essentially challenge Turing's thesis by examining a number of ways the standard Turing machine can be complicated to see if these complications increase its power. We look at a variety of different storage devices and even allow nondeterminism. None of these complications increases the essential power of the standard machine, which lends credibility to Turing's thesis. We look at several variations, showing that the standard Turing machine is equivalent to other, more complicated models.

To show that two models are equivalent, we simply need to show that one can simulate the other.

Part 2: A bilaterally infinite tape TM can be simulated by a unilaterally infinite tape TM

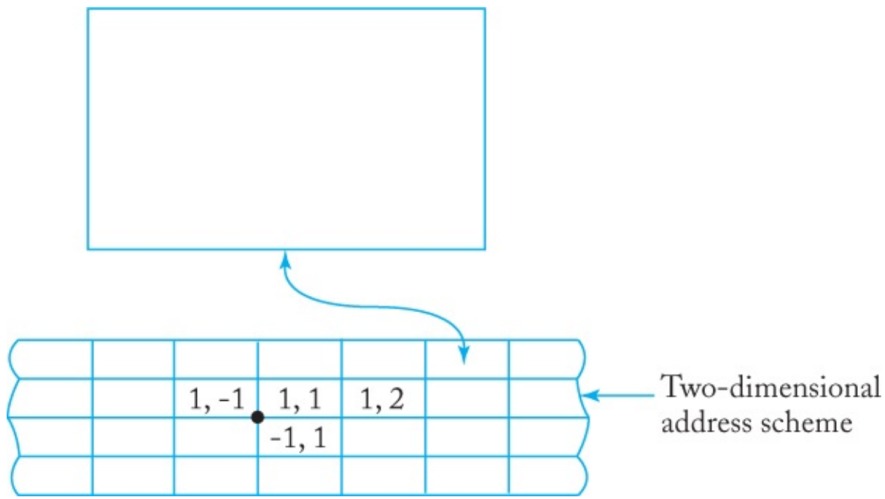
The Turing machine discussed so far has a tape infinite in the right direction, and is called a unilaterally infinite tape Turing machine. On the other hand, a Turing machine that has a tape infinite in both directions is called a bilaterally infinite tape Turing machine.

☞ *When given a Turing machine with an infinite tape in both directions, we can construct an equivalent Turing machine with a unilaterally infinite tape. This is described as the next theorem, whose proof is omitted.*

> ***Theorem:*** *For any bilaterally infinite tape Turing machine, there exists an equivalent unilaterally infinite tape Turing machine.*

Part 3: Multidimensional Turing Machines

A multidimensional Turing machine is one in which the tape can be viewed as extending infinitely in more than one dimension.



The formal definition of a two-dimensional Turing machine involves a transition function δ of the form $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$, where U and D specify movement of the read-write head up and down, respectively. It can be simulated by a standard TM.

Part 4: Turing Machines with a Stay-Option

To illustrate the robustness of the TM model, let's vary the type of transition function permitted. In our definition, the transition function forces the head to move to the left or right after each step; the head may not simply stay put. Suppose that we had allowed the TM the ability to stay put. The transition function would then have the form δ: $Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, S\}$. *Might this feature allow Turing machines to recognize additional languages, thus adding to the power of the model? Of course not, because we can convert any TM with the "stay put" feature to one that does not have it. We do so by replacing each stay put transition with two transitions: one that moves to the right and the second back to the left.*

This option does not extend the power of the automaton.

Theorem: *The class of Turing machines with a stay-option is equivalent to the class of standard Turing machines.*

Proof: *Since a Turing machine with a stay-option is clearly an extension of the standard model, it is obvious that any standard Turing machine can be simulated by one with a stay-option. To show the converse, let* $M = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$ *be a Turing machine with a stay-option to be simulated by a standard Turing machine* $\widehat{M} = (\widehat{Q}, \Sigma, \Gamma, \widehat{\delta}, \widehat{q_1}, \widehat{q_{\text{accept}}}, \widehat{q_{\text{reject}}})$. *For each move of* M*, the simulating machine* $\widehat{M}$ *does the following. If the move of* M *does not involve the stay-option, the simulating machine performs one move, essentially identical to the move to be simulated. If* S *is involved in the move of* M*, then* $\widehat{M}$ *will make two moves: The first rewrites the symbol and moves the read-write head right; the second moves the read-write head left, leaving the tape contents unaltered. The simulating machine can be constructed from* M *by defining* $\widehat{\delta}$ *as follows: For each transition* $\delta(q_i, a) = (q_j, b, L \text{ or } R)$,

we put into $\widehat{\delta}$

$$\widehat{\delta}(\widehat{q_i}, a) = (\widehat{q_j}, b, L \text{ or } R).$$

For each S*-transition* $\delta(q_i, a) = (q_j, b, S)$*, we put into* $\widehat{\delta}$ *the corresponding transitions*

$$\widehat{\delta}(\widehat{q_i}, a) = (\widehat{q_{jS}}, b, R),$$

and

$$\widehat{\delta}(\widehat{q_{jS}}, c) = (\widehat{q_j}, c, L),$$

for all $c \in \Gamma$*. It is reasonably obvious that every computation of* M *has a corresponding computation of* $\widehat{M}$*, so that* $\widehat{M}$ *can simulate* M*.*

Part 5: Multitape Turing Machine

(Simulating Multitape TM by Single-Tape TM)

A multitape Turing machine is like an ordinary Turing machine with several tapes. Each tape has its own head for reading and writing. Initially the input appears on tape 1, and the others start out blank. The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously. Formally, it is

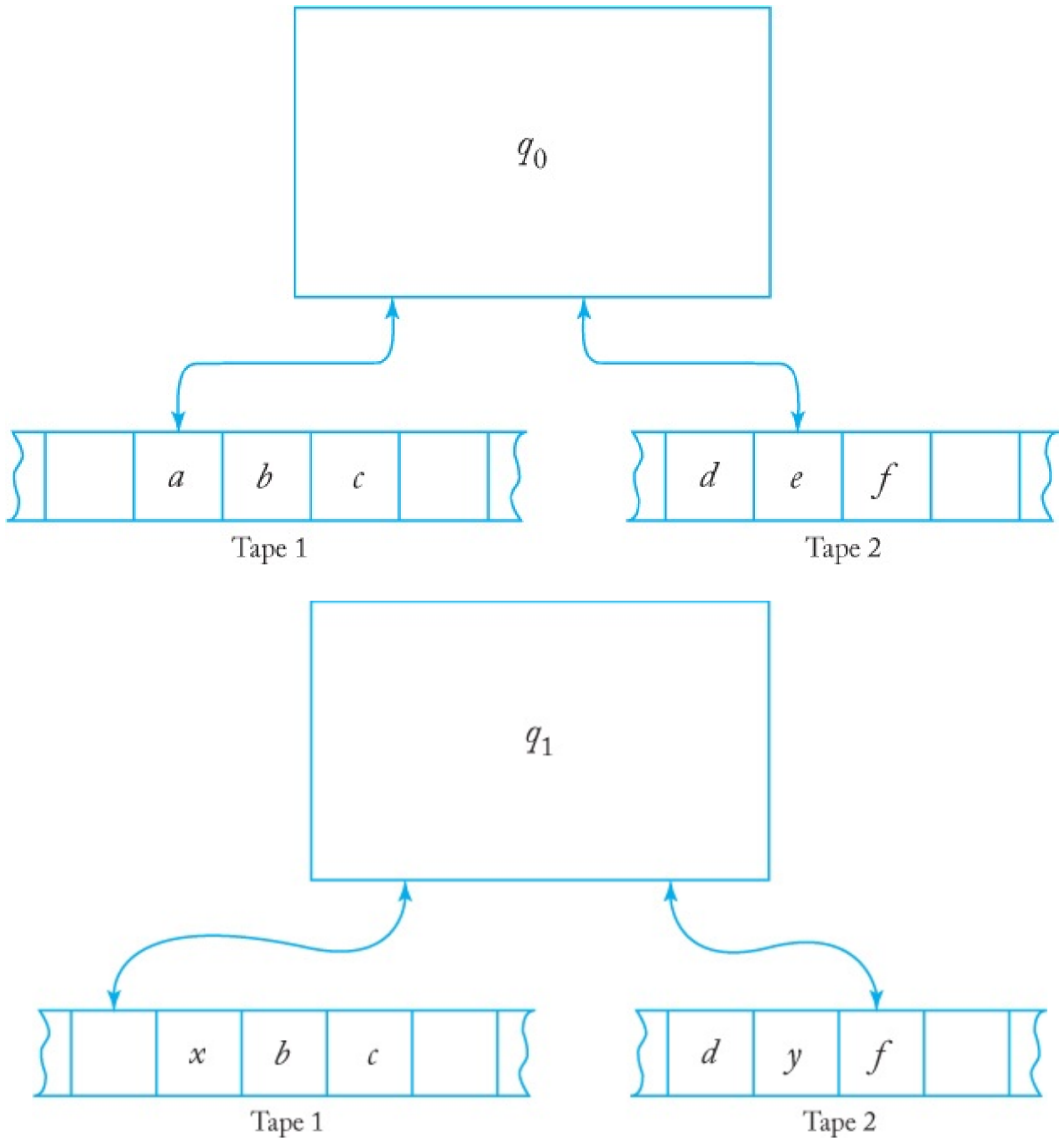
$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R, S\}^k,$$

where k is the number of tapes. The expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$ means that if the machine is in state q_i and heads 1 through k are reading symbols a_1 through a_k, the machine goes to state q_j, writes symbols b_1 through b_k, and directs each head to move left or right, or to stay put, as specified.

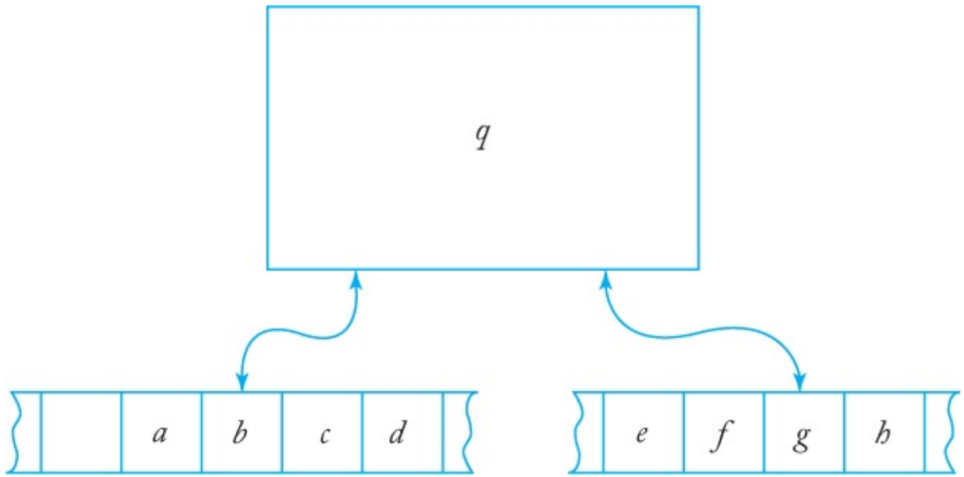
Since a multitape Turing machine has several tapes, it would seem to be more powerful than a single-tape Turing machine. It turns out, however, that they are equivalent in power to accept languages, which will be proved in this session. To prove this, it suffices to show that, given a k-tape Turing machine M, we can construct an equivalent single-tape Turing machine $\widehat{M}$. Recall that two machines are equivalent if they recognize the same language.

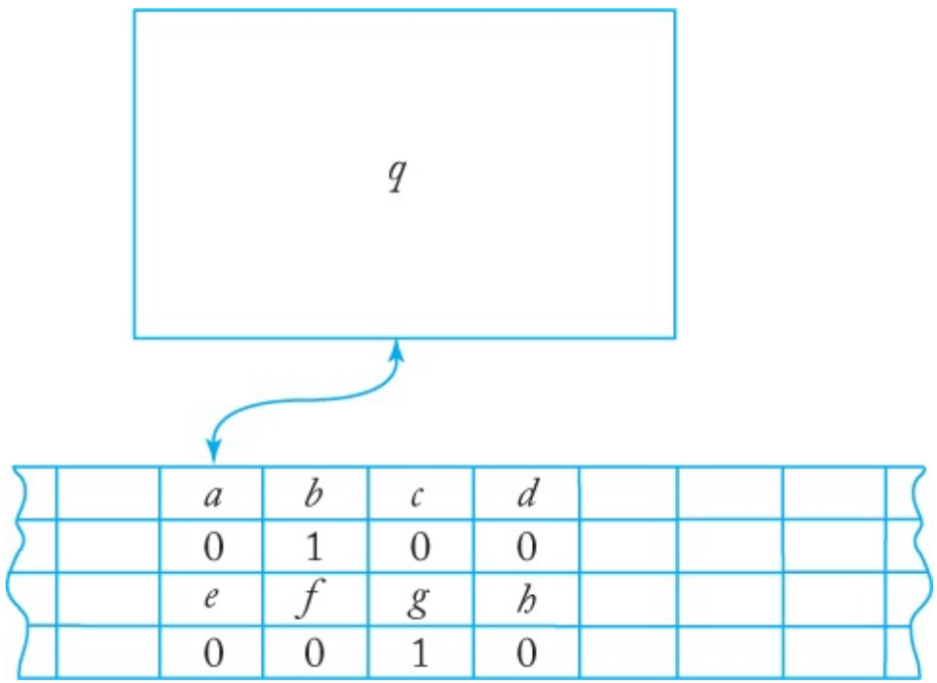
ما برای اثبات معادل بودن، دو اثبات متفاوت بیان می‌کنیم.

Example: If we have $\delta(q_0, a, e) = (q_1, x, y, L, R)$.



We argue that any given multitape TM M can be simulated by a standard TM $\widehat{M}$ and, conversely, that any standard TM can be simulated by a multitape one. The second part of this claim needs no elaboration, since we can always elect to run a multitape machine with only one of its tapes doing useful work. The simulation of a multitape machine by one with a single tape is a little more complicated, but conceptually straightforward:
Consider, for example, the two-tape machine in the configuration depicted in the following figure. The simulating single-tape machine will have four tracks. The first track represents the contents of tape 1 of M. The nonblank part of the second track has all zeros, except for a single 1 marking the position of M's read-write head. Tracks 3 and 4 play a similar role for tape 2 of M. The following figure makes it clear that, for the relevant configurations of $\widehat{M}$ (that is, the ones that have the indicated form), there is a unique corresponding configuration of M.

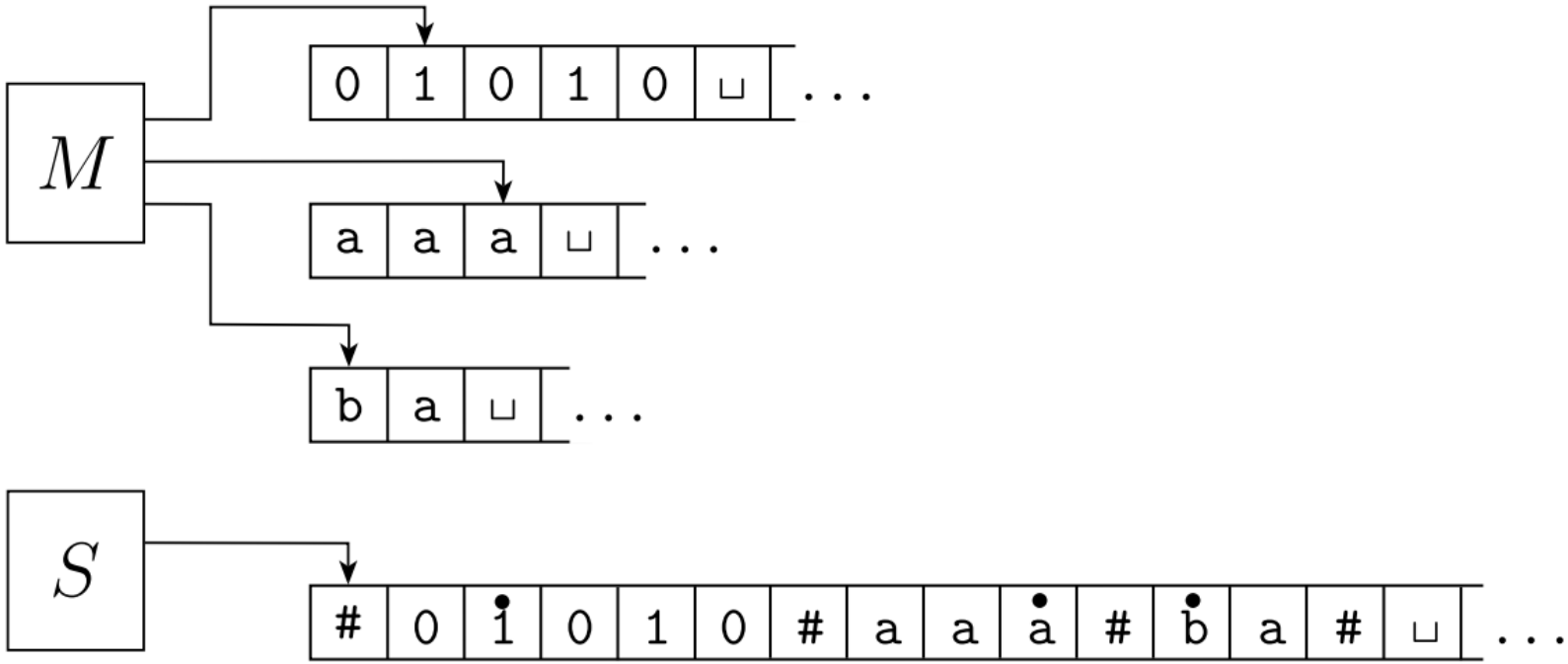

q
a
b
c
d
e
f
g
h


q
a
b
c
d
0
1
0
0
e
f
g
h
0
0
1
0

Remark: *In the definition of Turing machine, it is implicit that each tape symbol can be a composite of characters rather than just a single one. For example, in the above figure, the tape symbols are 4-tuples from some simpler alphabet. In the picture, we have divided each cell of the tape into four parts, called tracks, each containing one member of the 4-tuple. Based on this visualization, such an automaton is sometimes called a Turing machine with multiple tracks, but such a view is not an extension, since all we need to do is make* Γ *an alphabet in which each symbol is composed of several parts.*

Theorem: *Every multitape Turing machine has an equivalent single-tape Turing machine.*

Proof: *We show how to convert a multitape TM M to an equivalent singletape TM S. The key idea is to show how to simulate M with S. Say that M has k tapes. Then S simulates the effect of k tapes by storing their information on its single tape. It uses the new symbol # as a delimiter to separate the contents of the different tapes. In addition to the contents of these tapes, S must keep track of the locations of the heads. It does so by writing a tape symbol with a dot above it to mark the place where the head on that tape would be. Think of these as "virtual" tapes and heads. The "dotted" tape symbols are simply new symbols that have been added to the tape alphabet. The following figure illustrates how one tape can be used to represent three tapes.*



S = "**On input** $w = w_1 w_2 \cdots w_n$**:**

1. ***First*** S ***puts its tape into the format that represents all*** k ***tapes of*** M***. The formatted tape contains***

$$\#\dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#.$$

2. To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the $(k+1)$st # , which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M's transition function dictates.

3. If at any point S moves one of the virtual heads to the right onto a #, this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before."

Corollary: *A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.*

Proof: *A Turing-recognizable language is recognized by an ordinary (singletape) Turing machine, which is a special case of a multitape Turing machine. That proves one direction of this corollary. The other direction follows from the above theorem.*

Part 6: Nondeterministic Turing Machines

A nondeterministic Turing machine is defined in the expected way. At any point in a computation, the machine may proceed according to several possibilities. The transition function for a nondeterministic Turing machine has the form

$$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

The computation of a nondeterministic Turing machine is a tree whose branches correspond to different possibilities for the machine. If some branch of the computation leads to the accept state, the machine accepts its input. Now we show that nondeterminism does not affect the power of the Turing machine model. (Again we resort to simulation, showing that nondeterministic behavior can be handled deterministically.)

☞ *A nondeterministic Turing machine is allowed to choose the next move at any moment from among several possibilities.*

☞ *A transition function of a deterministic Turing machine takes the form*

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\},$$

whereas that of a nondeterministic one takes the form

$$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

☞ *A nondeterministic Turing machine is said to accept w if there is any possible sequence of moves such that $q_0 w \vdash^* x_1 q_{\text{accept}} x_2$.*

☞ *A typical specification of δ is given by*

$$\delta(q, a) = \{(q_1, a_1, D_1), \ldots, (q_m, a_m, D_m)\},$$

which means that there are m possibilities of the next move when the machine in state q reads symbol a.

☞ *A nondeterministic machine may have moves available that lead to a nonfinal state or to an infinite loop. But, as always with nondeterminism, these alternatives are irrelevant; all we are interested in is the existence of some sequence of moves leading to acceptance.*
Example: If a Turing machine has transitions specified by

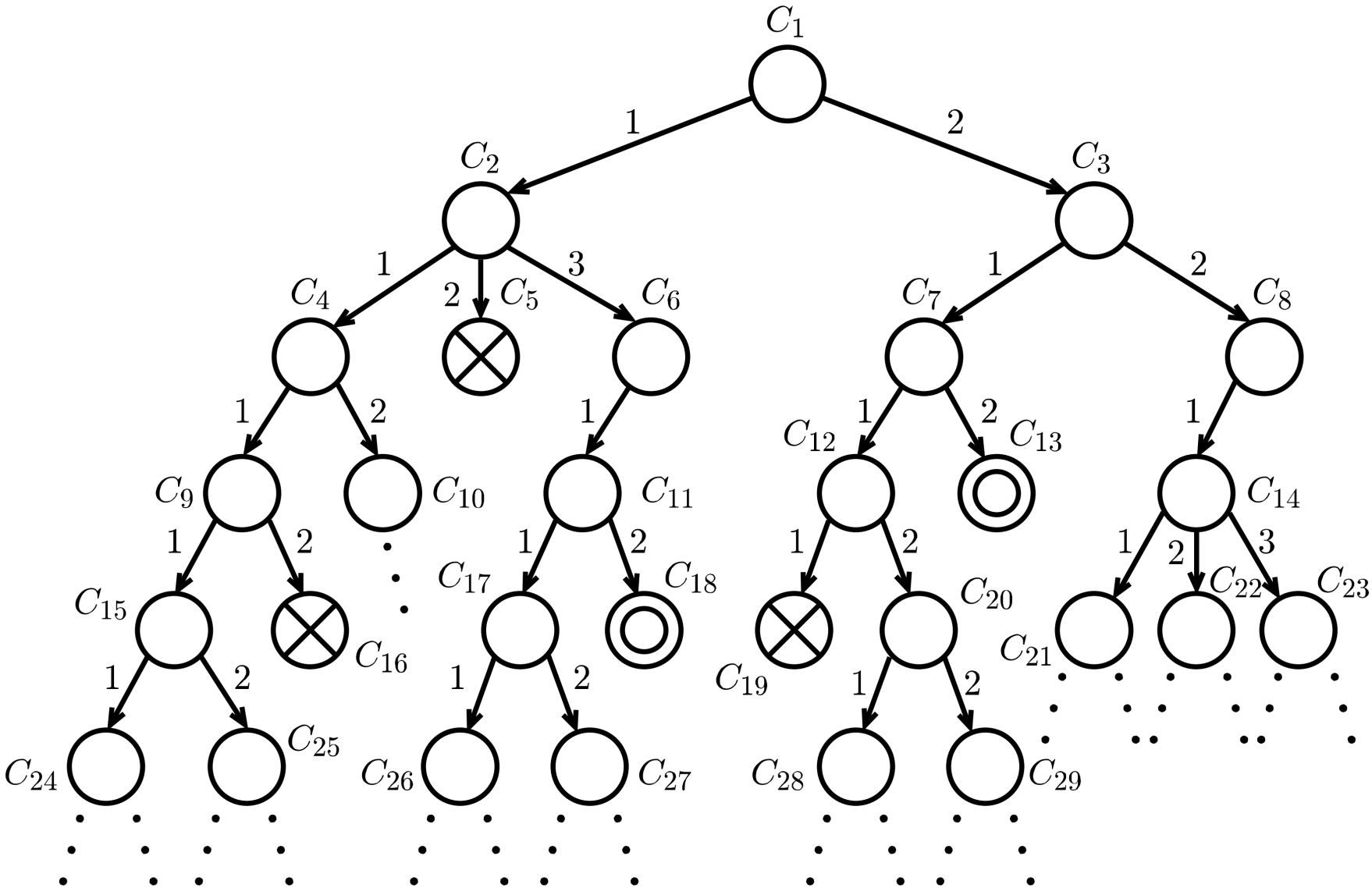
$$\delta(q_0, a) = \{(q_1, b, R), (q_2, c, L)\},$$

it is nondeterministic. The moves $q_0aaa \vdash bq_1aa$ *and* $q_0aaa \vdash q_2caa$ *are both possible.*

In our discussion a nondeterministic Turing machine will be abbreviated NTM, while a deterministic Turing machine will be abbreviated DTM.

For an NTM, each configuration has, in general, several subsequent configurations to which the former configuration moves. So, when given an NTM and its input, we can draw a tree which shows, beginning with the start configuration, how each configuration proceeds to subsequent configurations, splitting according to several possible transitions. Such a tree is called a computation tree.

Example of computation tree:



In this figure C_1 is the start configuration: if the start state and the input are denoted by q_1 and $a_1 \cdots a_n$, respectively, then C_1 is written $q_1 a_1 \ldots a_n$. In this case, since $\delta(q_1, a_1)$ is assumed to have two specifications, C_1 splits into two configurations C_2 and C_3. As in the case of state diagrams, C_{13} and C_{18} denote accepting configurations, whereas C_5, C_{16}, and C_{19} denote rejecting configurations.

An NTM M accepts string w if the computation tree based on M and w has at least one accepting configuration. On the other hand, M does not accept w if the computation tree does not have any accepting configuration. An NTM M accepts the language that consists of the strings that M accepts. As in the case of other computational models, the language that M accepts is denoted by $L(M)$.

In what follows, we shall construct a DTM D that simulates an arbitrarily given NTM N. The idea of how D works is that D goes around the entire computation tree defined based on NTM N and input w searching for an accepting configuration on the tree. If the computation tree has at least one accepting configuration, then D halts, accepting the input. On the other hand, if the tree does not have any accepting configuration, then D does not accept the input. If there exists no accepting configuration, there are two cases depending on whether the number of nodes in the computation tree is finite or infinite. In the finite case, D halts, rejecting the input. On the other hand, in the infinite case, D continues searching for an accepting configuration, but fails to find it, never halting to reject the input.

The crucial point in this argument is that we have to search the computation tree in such a way that if there exists an accepting configuration at all in the tree, D must eventually visit that configuration. If the condition described above is satisfied, DTM D is equivalent to NTM N in the sense that D accepts input w if and only if N accepts input w.

A tempting, though bad, idea is to have D explore the tree by using depth-first search. The depth-first search strategy goes all the way down one branch before backing up to explore other branches. If D were to explore the tree in this manner, D could go forever down one infinite branch and miss an accepting configuration on some other branch. Hence we design D to explore the tree by using breadth-first search instead. This strategy explores all branches to the same depth before going on to explore any branch to the next depth. This method guarantees that D will visit every node in the tree until it encounters an accepting configuration.

Now we proceed to explain in what order D visits the computation tree so that the condition described above is satisfied.

In order to represent a path in a computation tree by a sequence of integers, we assign integers to the possible transitions for each node in the tree. More precisely, let b denote the maximum of the numbers of possible transitions for the pairs expressed as (q, a)'s for state q and symbol a. Namely, let

$$b = \max_{q \in Q, a \in \Gamma} |\delta(q, a)|.$$

Furthermore, let us assume that for each pair (q, a) we assign integers 1 through j to the 3-tuples in $\delta(q, a)$, where $j = |\delta(q, a)| \leq b$. Then, obviously, a string u in $\{1, 2, \ldots, b\}^*$ specifies the path from the root to a node in the tree. So we can say such a string u indicates the node at the end point of the path.

☞ *To every node in the tree we assign an address that is a string over the alphabet $\{1, 2, \ldots, b\}$. Each symbol in the string tells us which choice to make next when simulating a step in one branch in N's nondeterministic computation.*

Of course, there might be a string in $\{1, \ldots, b\}^*$ that does not correspond to any path in the tree, and hence does not correspond to any node because some edges are lacking in the computation tree. (A symbol may not correspond to any choice if too few choices are available for a configuration. In that case, the address is invalid and doesn't correspond to any node.) While enumerating strings u in $\{1, 2, \ldots, b\}^*$ in lexicographical order DTM D visits the corresponding nodes in that order. In the course of the enumeration, if there exists no path that corresponds to string u or there exists a rejecting configuration on the path u, skip to the string that is lexicographically next to u.

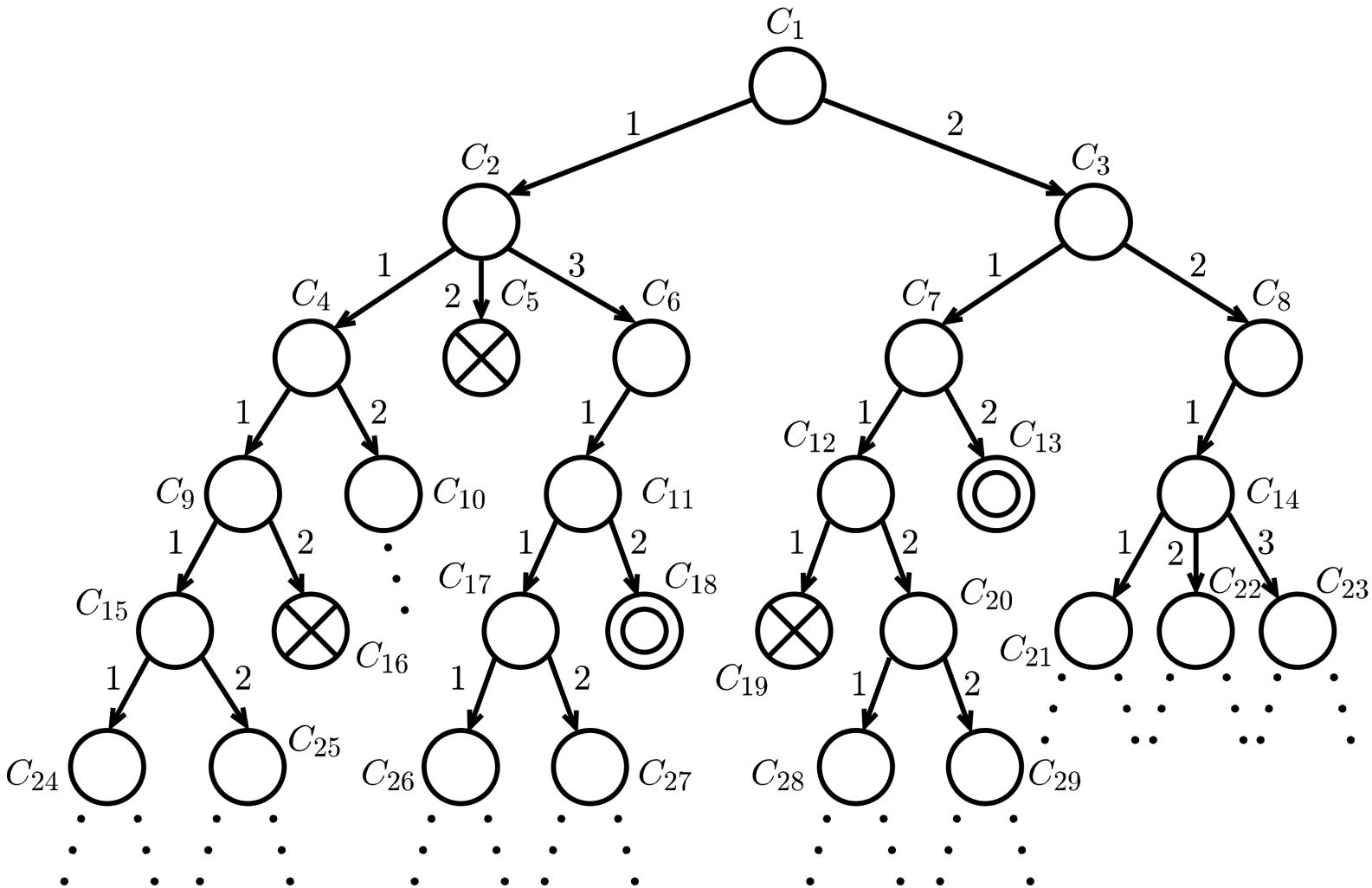
For example, in the case of $b = 3$, the lexicographical order of strings in $\{1, 2, 3\}^*$ is given by

$$\varepsilon, 1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, \ldots.$$

In the lexicographical order of strings shorter strings precede longer strings. Among strings of the same length the order is the same as the familiar dictionary ordering.

Given a string u in $\{1, 2, 3\}^*$, DTM D can simulate NTM N along the path associated with the string u on the computation tree. In simulating N this way, each digit of u tells us which choice to make next in the computation tree.

Example: ***In the following figure,*** 131 ***indicates*** C_{11}***, and*** 1312 ***indicates*** C_{18}***.*** 132 ***and*** 1313 ***correspond to no node.***



We summarize what is described so far in the next theorem.

Theorem: For any nondeterministic Turing machine, there exists an equivalent deterministic Turing machine.

Proof: We shall construct DTM D that is equivalent to given NTM N, as described before the theorem. Let

$$b = \max_{q \in Q, a \in \Gamma} |\delta(q, a)|.$$

Until an accepting configuration is encountered, DTM D repeats generating string u in $\{1, \ldots, b\}^*$ in lexicographical order to see if there exists an accepting configuration on the path u from the root in the computation tree by running NTM N choosing nondeterministic transitions according to u.

☞ *To do so, D uses three tapes: tape 1 is used to store input w throughout the computation; tape 2 is used as N's tape in simulating N along path u; tape 3 is used to generate string u in $\{1, \ldots, b\}^*$ in the lexicographical order.*

☞ *Tape 1 always contains the input string and is never altered.*

☞ *Tape 2 maintains a copy of N's tape on some branch of its nondeterministic computation.*

☞ *Tape 3 contains a string over $\{1, \ldots, b\}^*$. It represents the branch of N's computation from the root to the node addressed by that string unless the address is invalid. The empty string is the address of the root of the tree. (Tape 3 keeps track of D's location in N's nondeterministic computation tree.)*

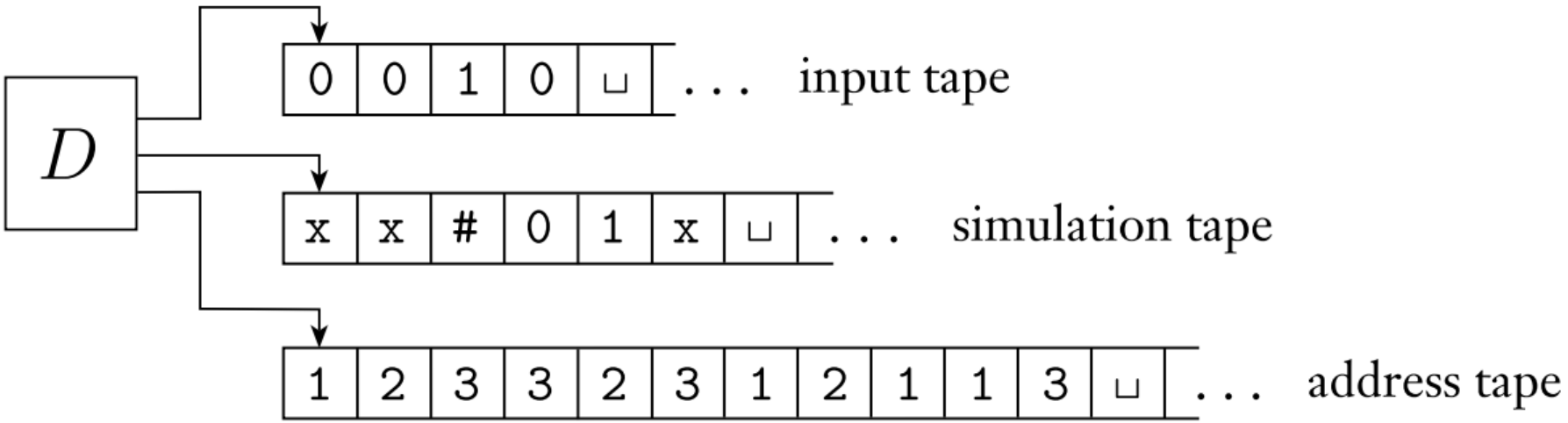
What is described informally so far is summarized as follows:

Three-tape DTM D that simulates NTM N:

1. *Place the input w in tape 1, make tape 2 empty, and place the empty string in tape 3.*
2. *Copy the input w to tape 2.*
3. *Let u be the string in tape 3. Simulate N with input w by choosing the nondeterministic transitions from the starting configuration successively according to u. If there exists no path that corresponds to u or a rejecting configuration is encountered in the course of the simulation, go to 4. If an accepting configuration is encountered, accept the input.*
4. *Replace the string on tape 3 with the string lexicographically next to the present string u and go to 2.*

Finally, there exists a single-tape deterministic TM that is equivalent to the DTM D with three tapes, completing the proof.

Deterministic TM D simulating nondeterministic TM N



Corollary: *A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.*

Proof: *Any deterministic TM is automatically a nondeterministic TM, and so one direction of this corollary follows immediately. The other direction follows from the above theorem.*

Part 7: Other theoretical models of computation

Other theoretical models of computation have been proposed. These include abstract machines such as the ones mentioned below, with two stacks or with a queue, as well as machines that are more like modern computers. In every case, the model has been shown to be equivalent to the Turing machine.

Problem 3.9: *Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.*

a. Show that 2-PDAs are more powerful than 1-PDAs.

b. Show that 3-PDAs are not more powerful than 2-PDAs. (Hint: Simulate a Turing machine tape with two stacks.)

Problem 3.14: *A queue automaton is like a push-down automaton except that the stack is replaced by a queue. A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we'll call it a push) adds a symbol to the left-hand end of the queue and each read operation (we'll call it a pull) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.*

تمرین: فرض کنید که برای ماشین تورینگ غیرقطعی N داریم $b = 5$. (یعنی حداکثر تعداد فرزندان یک نود در درخت محاسبه برابر با پنج است.) اگر قرار باشد که عملکرد ماشین N را به شیوهٔ مشروح در اسلایدهای قبل با یک ماشین تورینگ قطعی با ۳ نوار شبیه‌سازی کنیم، آنگاه رشته‌ای که بعد از رشتهٔ 324555555 روی نوار سوم ظاهر می‌شود چه رشته‌ای است؟ رشته‌ای که قبل از رشتهٔ 552311111 روی نوار سوم ظاهر می‌شود چه رشته‌ای است؟ محتوای نوار اول، حین فرایند محاسبه به چه نحو تغییر می‌کند؟