Finite State Machines 3

15-121 Introduction to Data Structures

Notes taken with modifications from "Introduction to Automata Theory, Languages, and Computation" by John Hopcroft and Jeffrey Ullman, 1979

Deterministic Finite-State Automata (review)

A DFSA can be formally defined as $M = (Q, \Sigma, \delta, q_0, F)$:
Q, a finite set of states
Σ, the alphabet of input symbols
δ, $Q \times \Sigma \rightarrow Q$, a transition function
q_0, the initial state
F, the set of final states

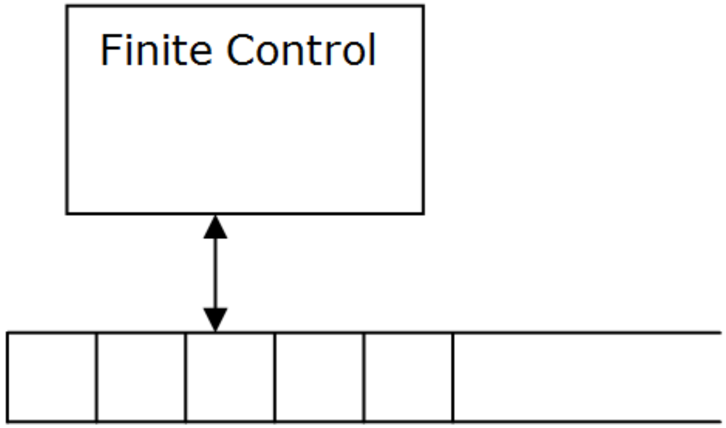
Pushdown Automata(review)

A pushdown automaton can be formally defined as M = $(Q,\Sigma, \Gamma, \delta, q_0, F)$:
Q, a finite set of states
Σ, the alphabet of tape symbols
Γ, the alphabet of stack symbols
δ, $Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma$
$q_{0,}$ the initial state
F, the set of final states

Carnegie Mellon

Turing Machines

- The basic model of a Turing machine has a finite control, an input tape that is divided into cells, and a tape head that scans one cell of the tape at a time.
- The tape has a leftmost cell but is infinite to the right.
- Each cell of the tape may hold exactly one of a finite number of tape symbols.
- Initially, the n leftmost cells, for some finite n >= 0, hold the input, which is a string of symbols chosen from a subset of the tape symbols called the input symbols.
- The remaining infinity of cells each hold the blank, which is a special symbol that is not an input symbol.



A Turing machine can be formally defined as M= (Q, Σ, Γ, δ, q_0 , B, F):
where
Q, a finite set of states
Γ, is the finite set of allowable tape symbols
B, a symbol from Γ is the blank
Σ, a subset of Γ not including B, is the set of input symbols
δ, $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ (may be undefined for some arguments)
q_0 in Q is the initial state
$F \subseteq Q$ is the set of final states

Turing Machine Example

The design of a Turing Machine M to decide the language L = {0^n1^n, n >= 1}. This language is decidable.

- Initially, the tape of M contains 0^n1^n followed by an infinity of blanks.
- Repeatedly, M replaces the leftmost 0 by X, moves right to the leftmost 1, replacing it by Y, moves left to find the rightmost X, then moves one cell right to the leftmost 0 and repeats the cycle.
- If, however, when searching for a 1, M finds a blank instead, then M halts without accepting. If, after changing a 1 to a Y, M finds no more 0's, then M checks that no more 1's remain, accepting if there are none.

Let $Q = \{ q_0, q_1, q_2, q_3, q_4 \}$, $\sum = \{0,1\}$, $\Gamma = \{0,1,X,Y,B\}$ and $F = \{q_4\}$
δ is defined with the following table:

INPUT SYMBOL

STATE	0	1	X	Y	B
q0	(q1,X,R)	-	-	(q3,Y,R)	-
q1	(q1,0,R)	(q2,Y,L)	-	(q1,Y,R)	-
q2	(q2,0,L)	-	(q0,X,R)	(q2,Y,L)	-
q3	-	-	-	(q3,Y,R)	(q4,B,R)
q4	-	-	-	-	-

As an exercise, draw a state diagram of this machine and trace its execution through 0011, 001101 and 001.

The Turing Machine as a computer of integer functions

- In addition to being a language acceptor, the Turing machine may be viewed as a computer of functions from integers to integers.
- The traditional approach is to represent integers in unary; the integer i >= 0 is represented by the string 0^i.
- If a function has more than one argument then the arguments may be placed on the tape separated by 1's.

For example, proper subtraction m – n is defined to be

m – n for m >= n, and

zero for m < n.

The TM M = ({q0,q1,...,q6}, {0,1}, {0,1,B}, δ, q0, B, {})

defined below, if started with 0^m10^n on its tape, halts with 0^{m-n} on its tape. M repeatedly replaces its leading 0 by blank, then searches right for a 1 followed by a 0 and changes the 0 to a 1. Next, M moves left until it encounters a blank and then repeats the cycle. The repetition ends if

- Searching right for a 0, M encounters a blank. Then, the n 0's in 0^m10^n have all been changed to 1's, and n+1 of the m 0's have been changed to B. M replaces the n+1 1's by a 0 and n B's, leaving m-n 0's on its tape.
- Beginning the cycle, M cannot find a 0 to change to a blank, because the first m 0's already have been changed. Then n >= m, so m – n = 0. M replaces all remaning 1's and 0's by B.

Carnegie Mellon

The function δ is described below.

δ(q0,0) = (q1,B,R) Begin. Replace the leading 0 by B.

δ(q1,0) = (q1,0,R) Search right looking for the first 1.
δ(q1,1) = (q2,1,R)

δ(q2,1) = (q2,1,R) Search right past 1's until encountering a 0. Change that 0 to 1.
δ(q2,0) = (q3,1,L)

δ(q3,0) = (q3,0,L) Move left to a blank. Enter state q0 to repeat the cycle.
δ(q3,1) = (q3,1,L)
δ(q3,B) = (q0,B,R)

If in state q2 a B is encountered before a 0, we have situation i described above. Enter state q4 and move left, changing all 1's to B's until encountering a B. This B is changed back to a 0, state q6 is entered and M halts.

δ(q2,B) = (q4,B,L)
δ(q4,1) = (q4,B,L)
δ(q4,0) = (q4,0,L)
δ(q4,B) = (q6,0,R)

If in state q0 a 1 is encountered instead of a 0, the first block of 0's has been exhausted, as in situation (ii) above. M enters state q5 to erase the rest of the tape, then enters q6 and halts.

δ(q0,1) = (q5,B,R)
δ(q5,0) = (q5,B,R)
δ(q5,1) = (q5,B,R)
δ(q5,B) = (q6,B,R)

As an exercise, trace the execution of this machine using an input tape with the symbols 0010.

Modifications To The Basic Machine

- It can be shown that the following modifications do not improve on the computing power of the basic Turing machine shown above:
  - Two-way infinite tape
  - Multi-tape Turing machine with k tape heads and k tapes
  - Multidimensional, Multi-headed, RAM, etc., etc.,…
  - Nondeterministic Turing machine
  - Let's look at a Nondeterministic Turing Machine…

Nondeterministic Turing Machine (NTM)

- The transition function has the form:
- δ: Q x Γ → *P*(Q x Γ x {L, R})
- So, the domain is an ordered pair, e.g., (q_0,1).
- Q x Γ x {L, R} looks like { (q_0,1,R),(q_0,0,R),(q_0,1,L),...}.
- *P*(Q x Γ x {L, R}) is the power set.
- *P*(Q x Γ x {L, R}) looks like { {}, {(q_0,1,R)}, {(q_0,1,R),(q_0,0,R)},...}
- So, if we see a 1 while in q_0 we might have to perform several activities…

Computing using a NTM

- A tree corresponds to the different possibilities. If some branch leads to an accept state, the machine accepts. If all branches lead to a reject state, the machine rejects.
- Solve subset sum in linear time with NTM:
- Set A = {a,b,c} and sum = x. Is there a subset of A summing to x? Suppose A = {1,2}, x = 3.   /   \
- for each element e of A   1 no 1

    take paths with and without e   / \   /\

accept if any path sums to x   2 no 2 2 no 2

accept reject reject reject

Church-Turing Hypothesis

Notes taken from "The Turing Omnibus", A.K. Dewdney

- Try as one might, there seems to be no way to define a mechanism of any type that computes more than a Turing machine is capable of computing.
- Note: On the previous slide we answered an NP-Complete problem in linear time with a non-deterministic algorithm.
- Quiz? Why does this not violate the Church-Turing Hypothesis?
- With respect to computability, non-determinism does not add power.

The Halting Problem

Notes taken from "Algorithmics The Sprit of Computing" by D. Harel

Consider the following algorithm A:

```
while(x != 1) x = x – 2;
stop
```

Assuming that its legal input consists of the positive integers <1,2,3,...>,It is obvious that A halts precisely for odd inputs. This problem can be expressed as a language recognition problem. How?

Now, consider Algorithm B:

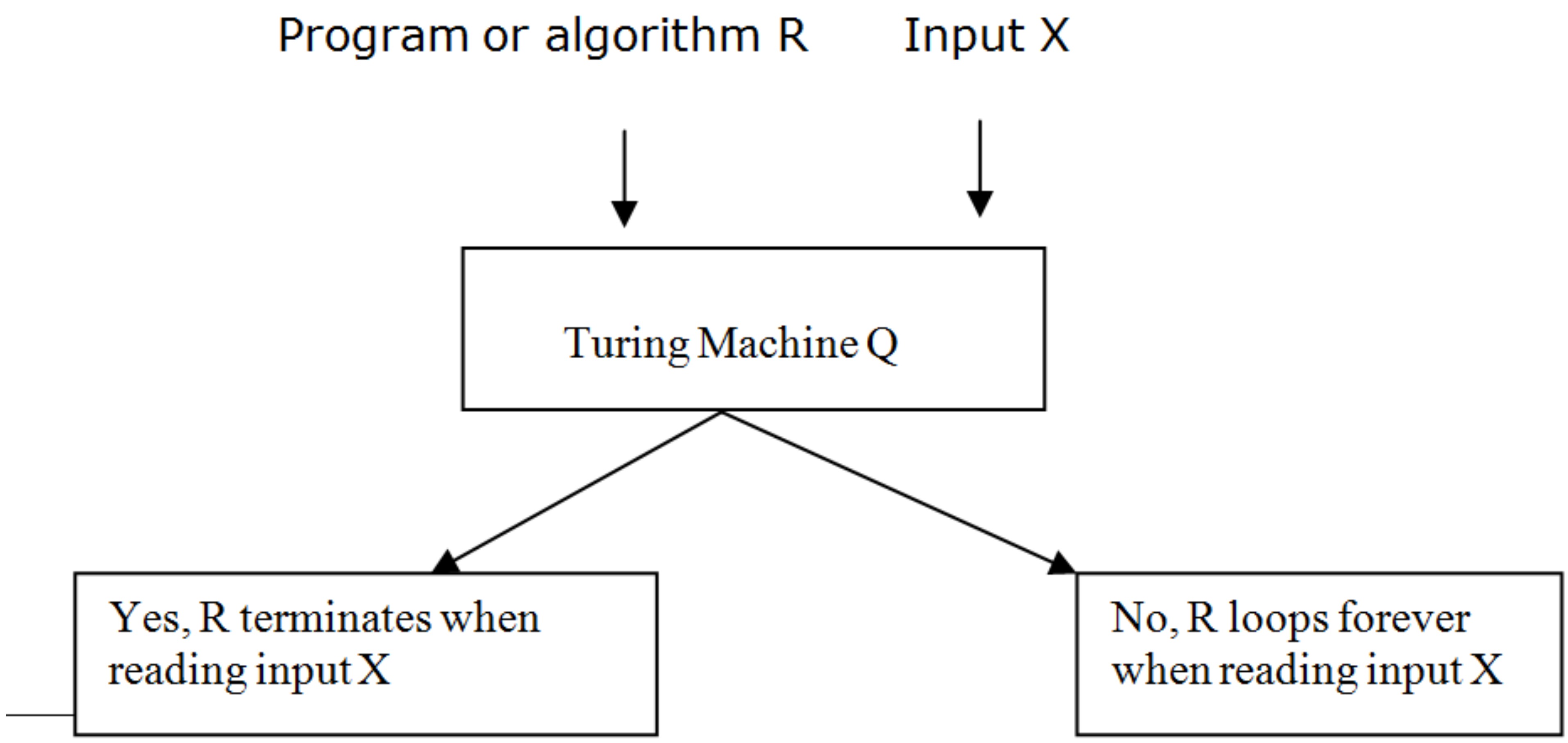
```
while (x != 1) {
        if (x % 2 == 0) x = x / 2;
          else x = 3 * x + 1;
  }
```

No one has been able to offer a proof that B always terminates. This is an open question in number theory. This too may be expressed as a language recognition problem.

The halting problem is "undecidable", meaning that there is no algorithm that will tell, in a finite amount of time, whether a given arbitrary program R, will terminate on a data input X or not.

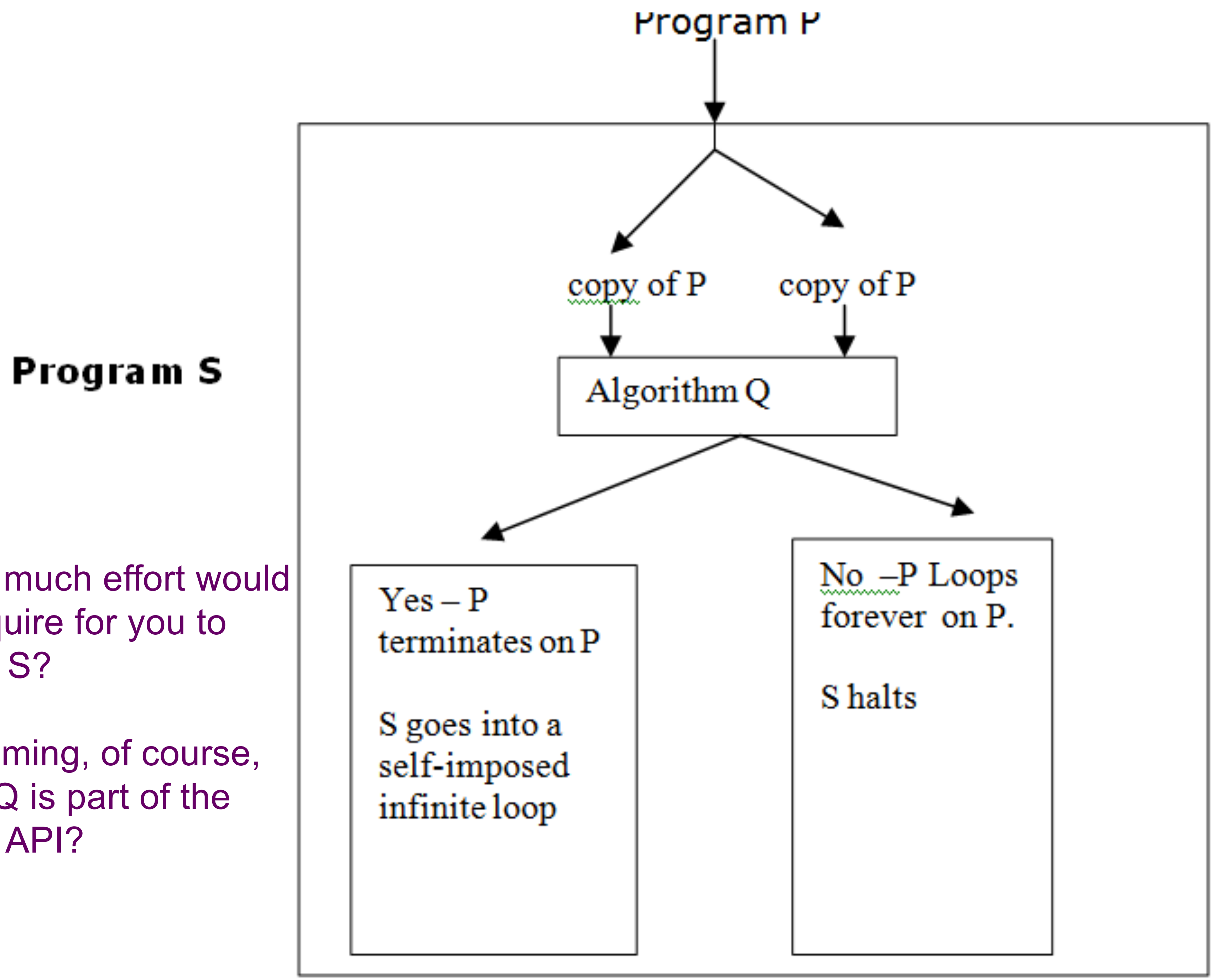
Carnegie Mellon

But let's build such a device anyway…



And let’s use it as a subroutine…

- Build a new program S that uses Q in the following way.
- S first makes a copy of its input. It then passes both copies (one as a program and another as its input) to Q.
- Q makes its decision as before and gives its result back to S.
- S halts if Q reports that Q’s input would loop forever.
- S itself loops forever if Q reports that Q’s input terminates.

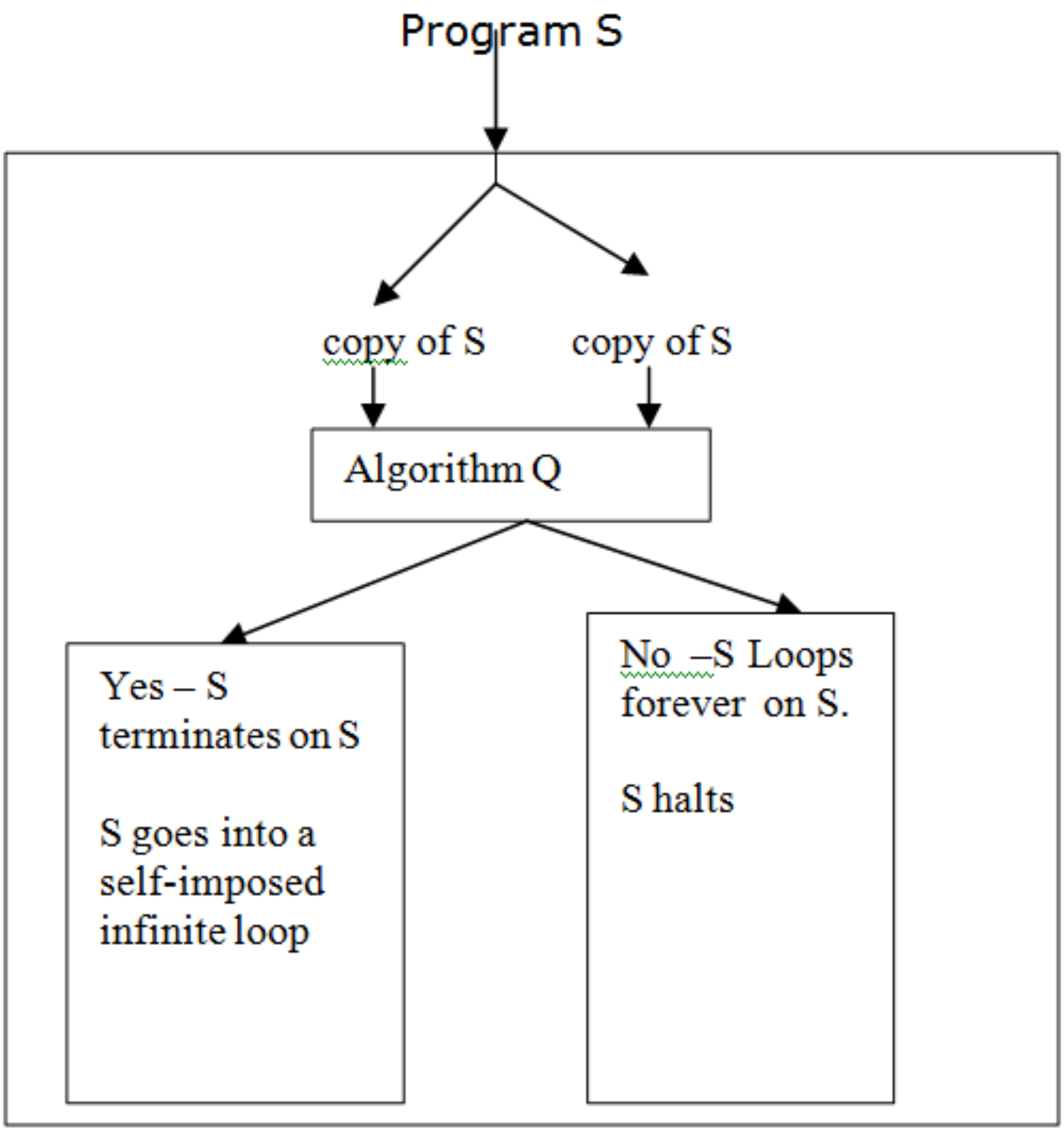


How much effort would It require for you to write S?

Assuming, of course, that Q is part of the Java API?

Program S

OK, so far so good. Now, pass S in to S as input.



- The existence of S leads to a logical contradiction. If S terminates when reading itself as input then Q reports this fact and S starts looping and never terminates. If S loops forever when reading itself as input then Q reports this to be the case and S terminates.
- The construction of S seems to be reasonable in many respects. It makes a copy of its input. It calls a function called Q. It gets a result back and uses that result to decide whether or not to loop (a bit strange but easy to program). So, the problem must be with Q. Its existence implies a contradiction. So, Q does not exist. The halting problem is undecidable.

Example: Malware Detection

- Shown to be undecidable
- Do we give up?
- No – monitoring output of processes can still be fruitful

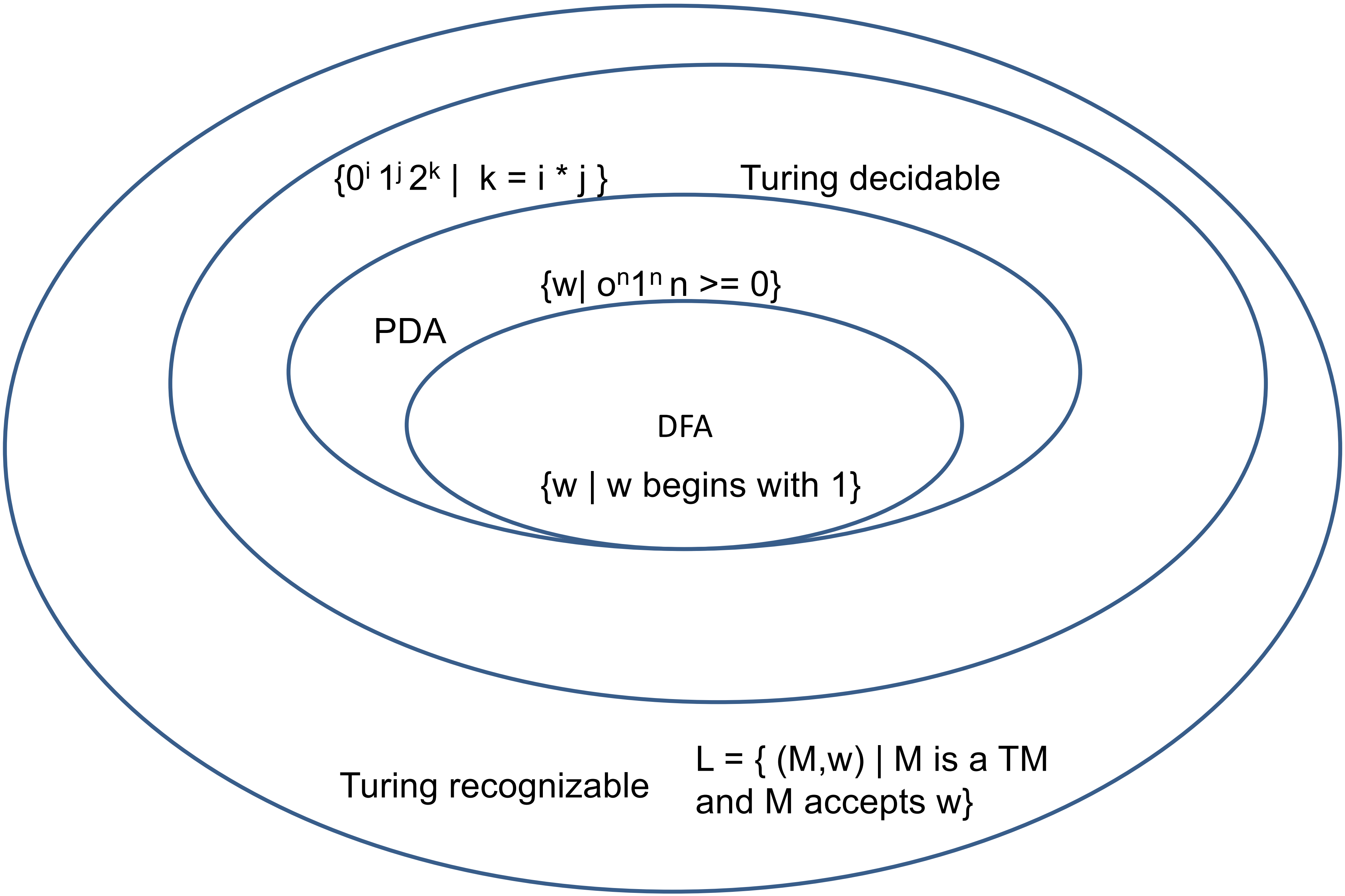
Terminology: Recursive and Recursively Enumerable notes from Wikipedia

- A formal language is ***recursive*** if there exists a Turing machine which halts for every given input and always either accepts or rejects candidate strings. This is also called a ***decidable*** language.
- A ***recursively enumerable*** language requires that some Turing machine halts and accepts when presented with a string in the language. It may either halt and reject or loop forever when presented with a string not in the language. A machine can ***recognize*** the language.
- The set of halting program integer pairs is in R.E. but is not recursive. We can't decide it but we can recognize it.
- All recursive (decidable) languages are recursively enumerable.

Recursive and Recursively Enumerable

- The set of halting program integer pairs is in R.E. but is not recursive.
- Are there any languages that are not recursively enumerable?
- Yes. Let L be { w = (program p, integer i) | p loops forever on i}.
- L is not recursively enumerable.
- We can't even recognize L.
- The set of languages is bigger than the set of Turing machines.

Carnegie Mellon



Co-L = {(M,w) | M is a TM and M rejects w or loops}

Some Results First

Computing Model	Finite Automata	Pushdown Automata	Linear Bounded Automata	Turing Machines
Language Class	Regular Languages	Context-Free Languages	Context-Sensitive Languages	Recursively Enumerable Languages
Non-determinism	Makes no difference	Makes a difference	No one knows	Makes no difference

26