(1) New TA: Jingyuan Fan Offc hrs Wed 3-5pm
KWR: 1-2 Tue, 3-5 Wed, 1 more hr TBA
James Clay: Mon 5-6 Fri 3-4pm

A language represents:
- A logical condition on a string
- A yes/no decision problem with an (arbitrary but) specific encoding of objects as strings.

Example 1: Given a positive number n, is n a difference of two powers of 2?

Instances: $n = 7$? Yes: $2^3 - 2^0 = 8 - 1 = 7$
$n = 120$? Yes: $2^7 - 2^3 = 128 - 8 = 120$
$n = 26$? No!
$n = 54$? No.
$n = 1,260$?
$n = 8$?

$7 = 111$
$120 = 1111000$
$54 = 110110$
$1260 = \sim$
$8 =$

Algorithm: The answer is yes unless a '1' follows a '0' in n, under standard binary encoding (note: $n > 0$).

I actually gave a logical assertion that implies an algorithm and states its correctness. Prove it?

"Proof by Picture": $2^k = 1\,0\,0\,0\,0\,0\,0\,0\,0 \ldots 0$
For some $j < k$, $2^j = 1\,0\,0\,0\,0 \ldots 0$

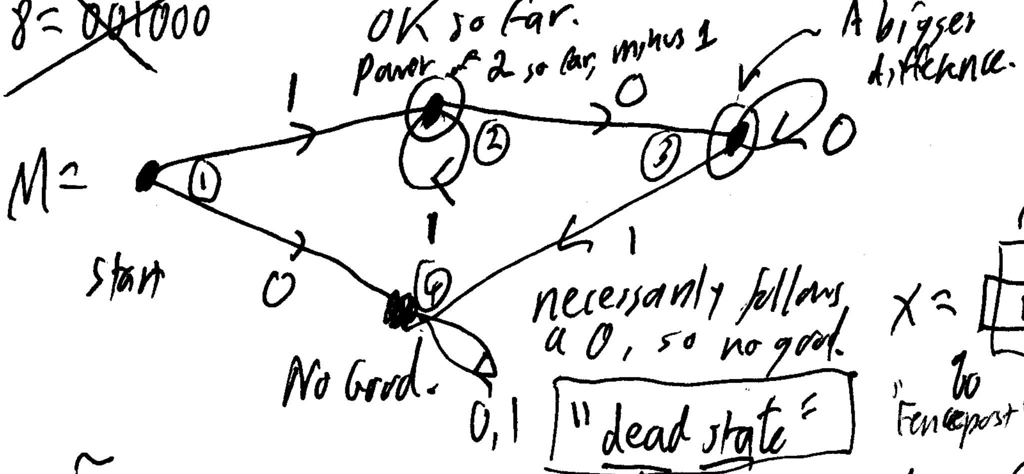
$1\,1\,1\,1\,0\,0\,0\,0 \ldots 0$

Output always has one or more 1s followed by zero or more 0s.
Shorthand: $\text{Binary}(n) = \underbrace{1 \cdot 1^*}_{A} \cdot \underbrace{0^*}_{B} = 1^+ 0^*$ superscript + means one-or-more

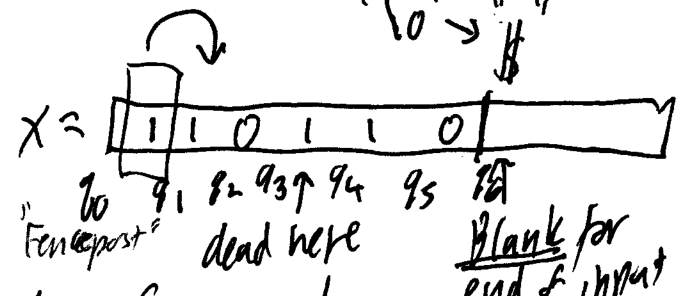
Part II: A machine-type algorithm for this problem.

Language $L = 1 \cdot 1^* \cdot 0^* = 1^+ 0^* = \{x \in \{0,1\}^* :$ x begins with at least one 1 and has 1s followed optionally by 0s $\}$.

$8 = \cancel{00}1000$

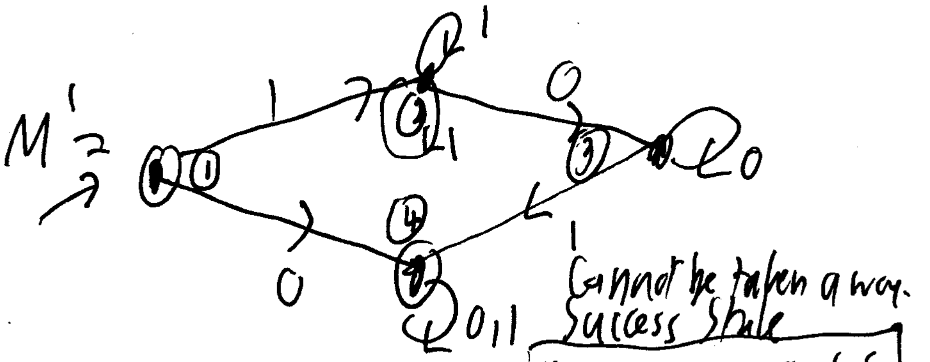


'10' → "#"



$\tilde{L} = \{x :$ x is a standard binary encoding of a number that is not a diff of two powers of 2, or x is not a legal encoding$\}$

x = ϵ or x has leading 0s.



"Nirvana State"

Desired Final States.

For M, $F = \{2, 3\}$

$Q = \{1, 2, 3, 4\}$

For M': $F' = \{1, 4\}$ instead.

Formal Definition 51.1:

A deterministic finite automaton (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, s, F)$ where:

- Q is a finite set of elements called states.
- Σ is the input alphabet.
- s, a member of Q, is the start state (text: q_0)
- F, a subset of Q, is the set of final states
- $\delta : Q \times \Sigma \to Q$.

Here: $\delta(1,0) = 4$ $\delta(1,1) = 2$
$\delta(2,0) = 3$ $\delta(2,1) = 2$
$\delta(3,0) = 3$ $\delta(3,1) = 4$
$\delta(4,0) = \delta(4,1) = 4$

$Q = \{1, 2, 3, 4\}$ $\Sigma = \{0,1\}$
$s = 1$ $F = \{1, 4\}$ in M'

≡ Given char and a State type already enum

```
class DFA {
   set<State> Q;
   set<char> Sigma;
   State s;
   set<State> F;
   State (*delta)(State p, char c);
        [Member function rather than a method]
   ~~State delta(State p, char c);~~
}
```

Segue to Thursday's lecture

Extra ③

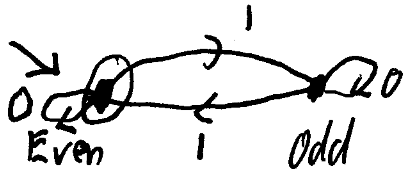
Note: 0 is an even number.

Example 2: $L_2 = \{x \in \{0,1\}^* : \text{the number of 1s in } x \text{ is even}\}$

abbreviated $\#1(x)$ or $N_1(x)$, not in text.

Unlike Example 1, I don't know a simple way to define the corresponding set of binary numbers "numerically".

The algorithm is easy though — a DFA with only 2 states will do:



$Q = \{\text{"Even"}, \text{"Odd"}\}$
$\Sigma = \{0,1\}$, $s = \text{"Even"}$
$F = \{\text{"Even"}\} = \{s\}$

$\delta(\text{Even},0) = \text{Even}$
$\delta(\text{Even},1) = \text{Odd}$
$\delta(\text{Odd},0) = \text{Odd}$
$\delta(\text{Odd},1) = \text{Even}$

Using the idea of the graph of a function, we can rewrite δ as a set of instructions:

Set3"

set<State, char, State> delta = {(Even,0,Even), (Even,1,Odd), (Odd,0,Odd), (Odd,1,Even)}

Note we defined the whole DFA without using numbers for states: State can be anything. To be super-technical we should write the graph form of e.g. $\delta(\text{Even},1) = \text{Odd}$ as a nested pair ((Even,1), Odd), but simple triples are A-OK. C++/Java/etc. let you do it either way too:

Set<Triple<State, char, State>> or set<Pair<Pair<State, char>, State> >

C++ does not have a standard Triple type. You could write your own "Set3" class to be like above. "Yuck!" (use typedefs?) in C++ need this space, Java and C# OK not

A regular expression for L_2 is easy too. We can reason directly or by "tracing" the DFA.

- Directly: '0' doesn't matter, so we can stick "zero or more 0s" anywhere. What we need is "zero or more pairs of 1s." By itself that's $(11)^*$. Combined gives $(0^*10^*10^*)^*$.

- From the machine: We need to begin and end at the "Even" state. We can come back in two ways: a single 0; or a 1 then any number of 0s then a 1 again. Expression:

$$\text{Comeback} = 0 \cup 10^*1.$$

We begin and need to do zero-or-more "Comeback"s. So we get: $(0 \overset{\text{or } \cup}{+} 10^*1)^*$

Hey! The answers are different! Is one wrong? No — but it's a sign of a "bumpy ride"...