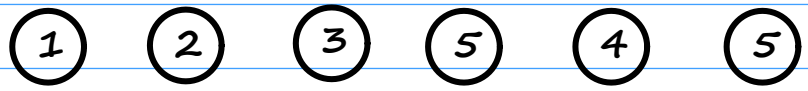
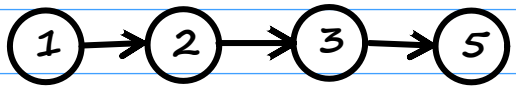
Before we talk about Tuco...

Let's say we have a list

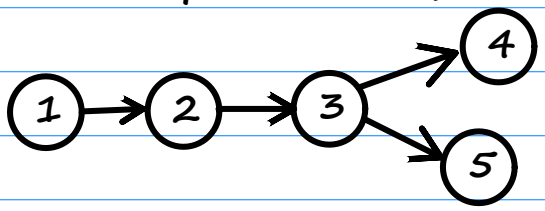


How would we solve it?

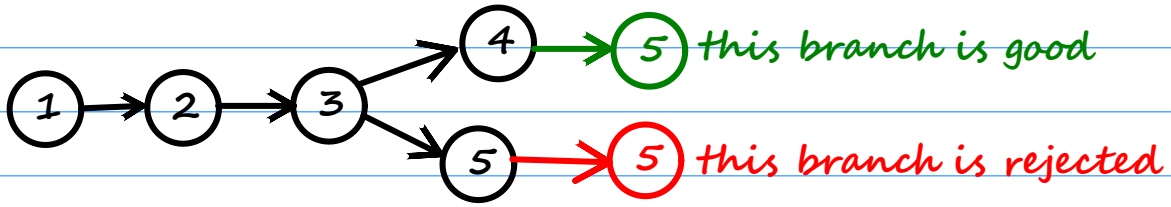
Up to 5 the difference is always within the [1, 3] bound



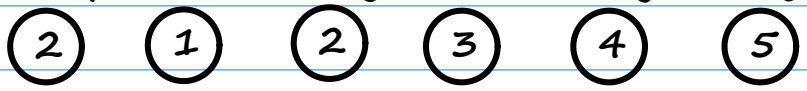
But 4 is problematic, we can either delete 4 or 5:



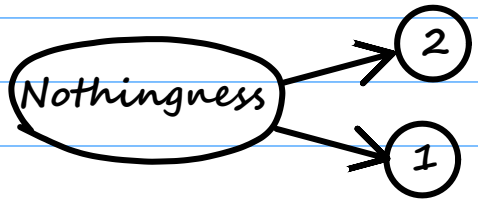
But now after this 'fork'...
we can no longer delete elements
we 'used' up our deletion



Another problem is right at the beginning



We have to 'fork' immediately



We have to represent the idea of Nothingness too

Can we somehow create a Machine, such that:
- we can feed Integers into it one by one
- and it just does state transitions magically
- and accepts / rejects?

Tuco Salamanca variant

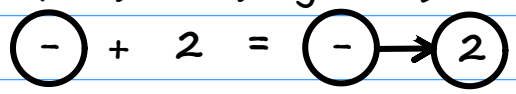
I'll use P to denote a predicate

$$P(x, y) = 1 <= (y - x) \text{ AND } (y - x) <= 3$$

We don't know what states are possible, let's start with Empty!

1) Empty

There is one single case:



The only reasonable action is to store 'a'

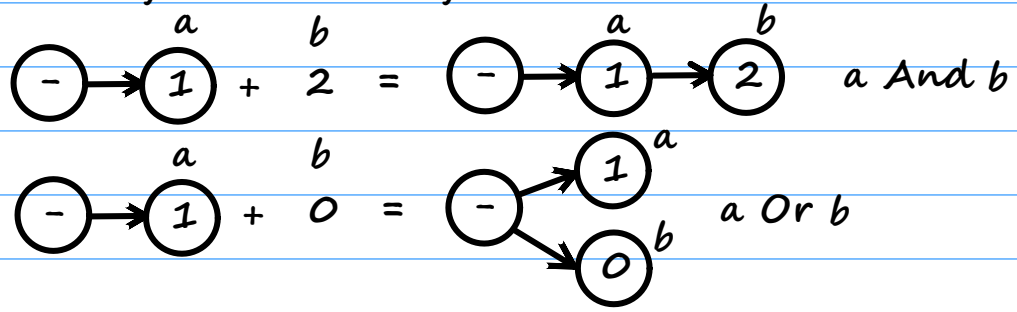
We will call this new state 'One'

Empty + a = One a

One is a new 'state' let's deal with it next!

2) One a

Think of what kind of cases do we bhave:



(One a) + b = if P a b then a And b else a Or b

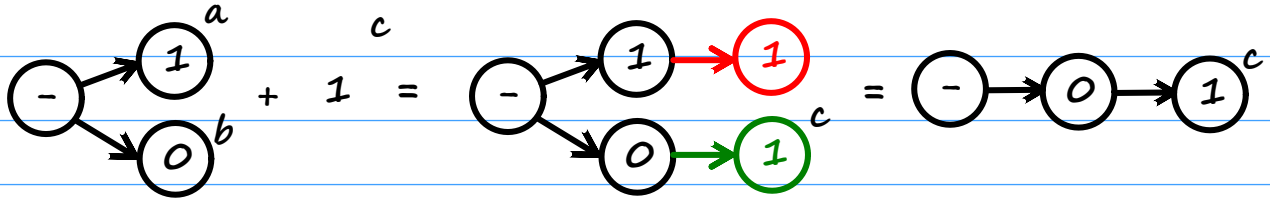
And and Or are new 'states' we'll have to deal next

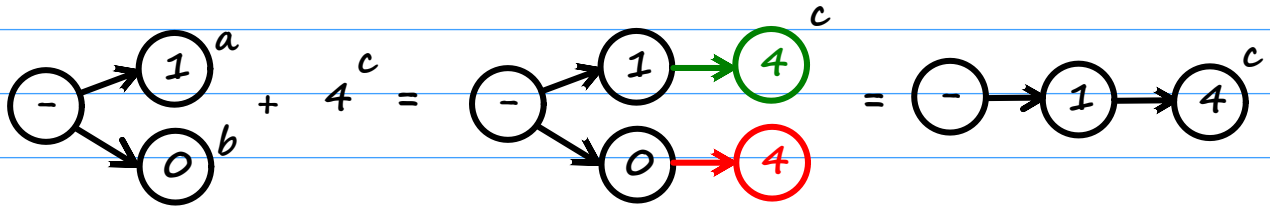
Let's start with Or!

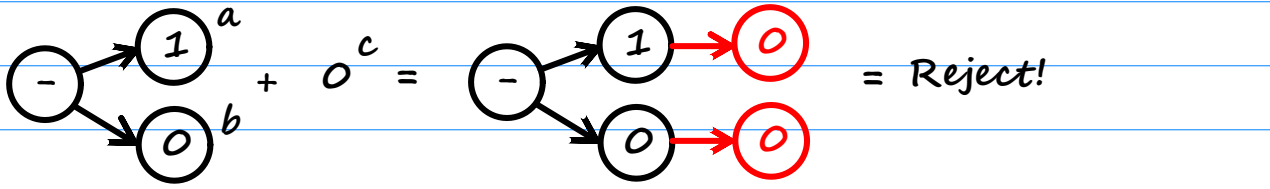
Tuco Salamanca variant

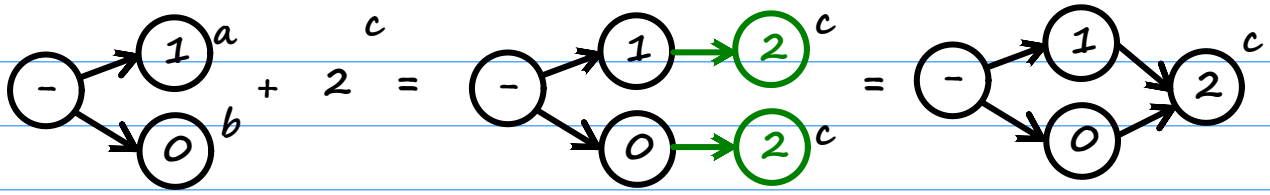
3) a Or b

Think of what kind of cases do we have:









(a Or b) + c = if P a c || P b c then Tail c else Reject

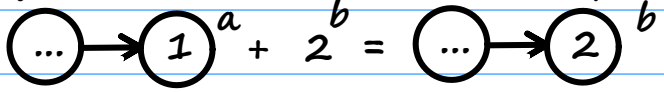
4) Reject

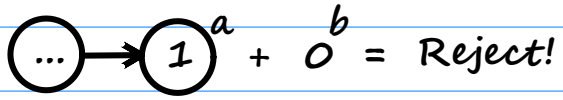
This Reject state is simple, it has one case:

Reject + a = Reject

5) Tail

If we are at Tail, we already used up the delete, so:



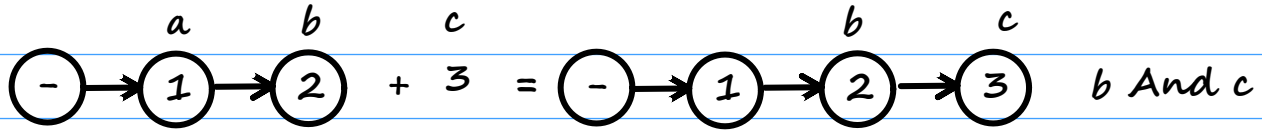


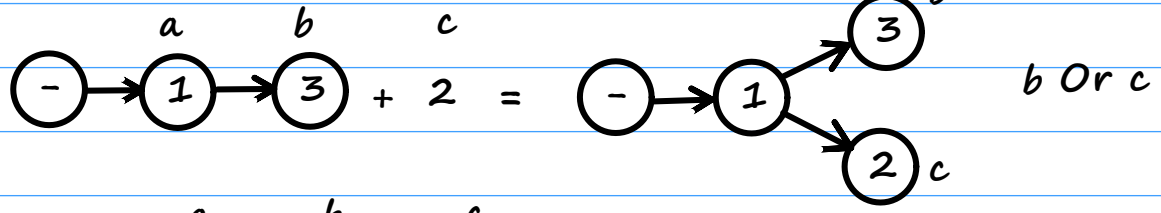
(Tail a) + b = if P a b then Tail b else Reject

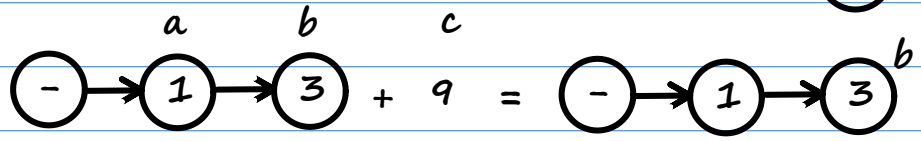
Tuco Salamanca variant

6) a And b

Think of what kind of cases do we have:







* we deleted '9'

```
(a And b) + c
    | P b c      = b And c
    | P a c      = b Or c
    | otherwise  = Tail b
```

Tuco Salamanca variant

Summary

The elements of the Set are

```
Empty | One a | a Or b | Reject | Tail a | a And b
```

The binary operation '+' closed over the Set is

```
Empty + a = One a

(One a) + b = if P a b then a And b
              else a Or b

(a Or b) + c = if P a c || P b c then Tail c
               else Reject

Reject + a = Reject

(Tail a) + b = if P a b then Tail b
               else Reject

(a And b) + c
    | P b c     = b And c
    | P a c     = b Or c
    | otherwise = Tail b
```

And for a list 'xs' we can solve it by:

```
let x0 = Empty
for (x : xs) { x0 = x0 + x }
return x0
```

And this is it. The (Set, +) is the Machine we wanted.

Rude, brutish, aggressive

but hammers the problem into oblivion

Just like Angular/React/SpringBoot/RubyOnRails...

https://youtu.be/lw9PhA413v1?t=156