The Machines of Computation

Talking in abstracts

- Abstract computation machines
- Operate on a **series of symbols** as **input**
- Produce a **series of symbols** as **output** or **make a decision**

Look-up table

- Just has a specified output for every possible input

a	**A**
A	**A**
b	**B**
B	**B**
c	**C**
C	**C**
d	**D**
D	**D**
...	**...**

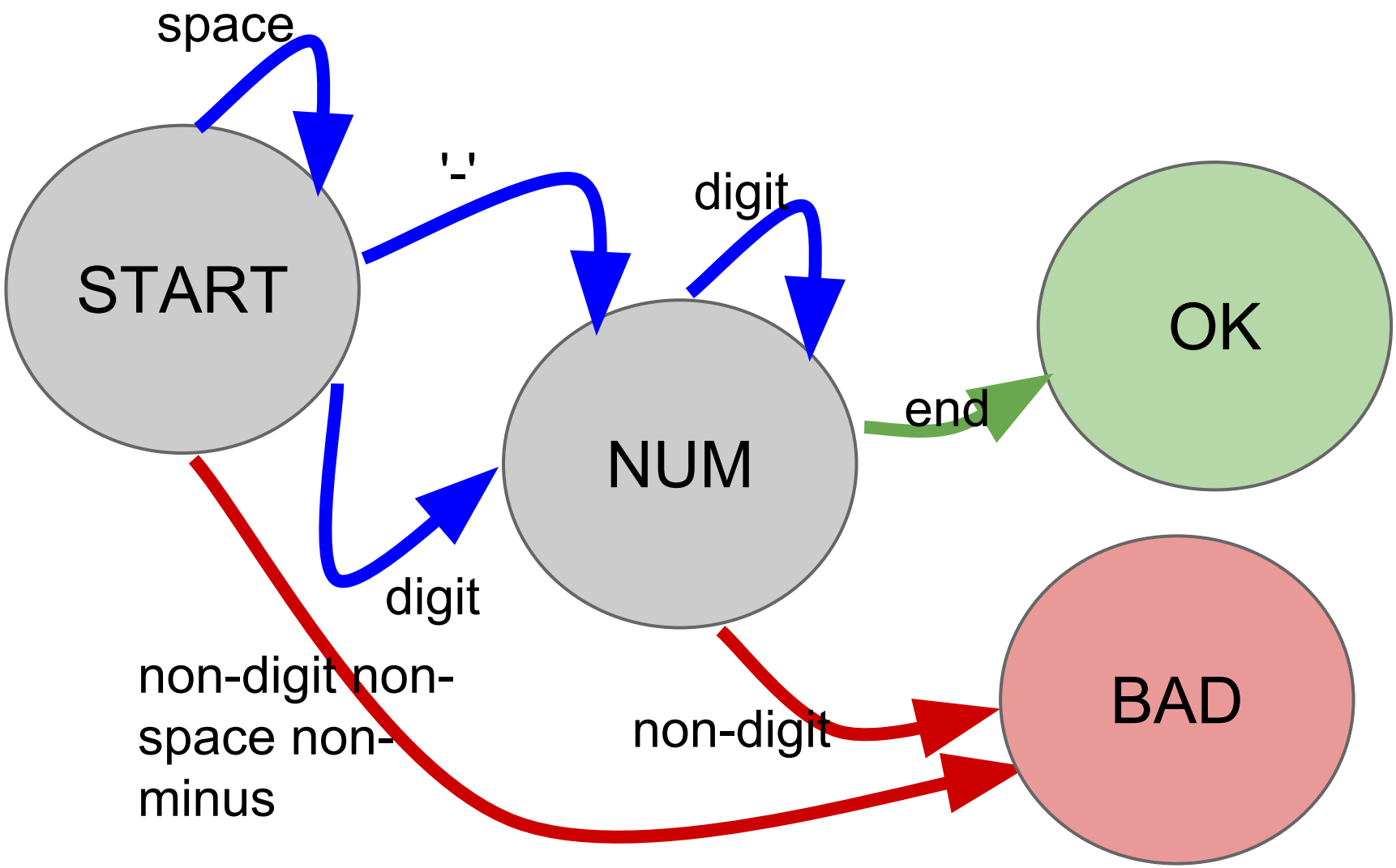
Finite State Machine

- Just like a look-up table
- ...except we also store a single piece of information, our **state**
- We have to be able to list out all valid states we could be in -- in other words, our allowable state has to be **finite**

Finite State Machine

- **recognizers** only care about the state they end up in
- **transducers** produce output at each step
- Now our table includes both the **input** and the **state** that we're looking up, and both the **output** (if we have it) and the **next state** that we're producing

Recognize valid numbers



Recognize valid numbers

Input	State	Next state
'-'	START	NUMBER
any digit	START	NUMBER
any space	START	START
not space, digit, or '-'	START	BAD
any digit	NUMBER	NUMBER
any non-digit	NUMBER	BAD
end of input	NUMBER	OK

Recognizers and Languages

- Divide the world of possible input sequences into **accepted** and **rejected** sequences
- The set of all sequences that would be accepted are considered to be a "**language**"
- Languages that can be recognized by a finite state machine are called **regular languages**
- It turns out regular languages are pretty useful. In fact, we've developed a shorthand for defining whole recognizer FSMs in one go.

Regular Expressions

- **any letter**: recognizes that letter
- **star ('*')**: recognizes the preceding item any number of times
- **question ('?')**: recognizes the preceding item 0 or 1 time

These are going to look cryptic, but take them one character at a time.

ab?c*

Regular Expressions

- **any letter**: recognizes that letter
- **dot ('.')**: recognizes any character
- **parentheses**: makes a group
- **star ('*')**: recognizes the preceding item any number of times
- **plus ('+')**: recognizes the preceding item at least once but maybe more times
- **question ('?')**: recognizes the preceding item 0 or 1 time
- **brackets ([])**: recognizes any one of the things in the brackets. You can specify a range of characters with '-'.

Regular Expressions

- **\s**: any space
- **\w**: any "word character". Same as [a-zA-Z0-9_]
- **\d**: any digit. Same as [0-9]
- **^**: the beginning of a string
- **$**: the end of a string
- **For special characters, put a backslash in front of it to mean the character itself**

.*\.py

^...B.*

```
^\s*-?\d+\s*$
```

In Python

```
import re

m = re.match("^...B.*", "BLABS")

m = re.match("^...B.*", "BLUE")
```

Finite state machines can only do so much

- Limited by their number of states
- Can't handle things like nesting
- The classic language that can't be recognized by a finite state machine: correctly nested parentheses

Let's equip our finite state machine with an infinite paper tape

- Instead of input, reads a symbol from the tape
- Instead of output, writes a symbol to the tape, replacing whatever was there before
- Add to the FSM table: Go **L**eft, **R**ight, or **N**owhere on the tape.

http://morphett.info/turing/turing.html

Turing Equivalence

Machines A and B are **Turing equivalent** if:

- You can simulate A using B
- You can simulate B using A

The Church-Turing Thesis

- A Turing machine (a FSM with an infinite tape) is the most powerful kind of computer known
- That doesn't mean it's particularly powerful, but rather that everything you can compute, you can compute with a Turing machine
- A Turing machine is Turing-equivalent to pretty much every programming language

Everything you can compute...

What can't you compute?

The Halting Problem

```
def halts(function, argument):
    """Returns True if the function
    ever exits when it is applied to
    argument, False otherwise"""
    .... can we write this?
```

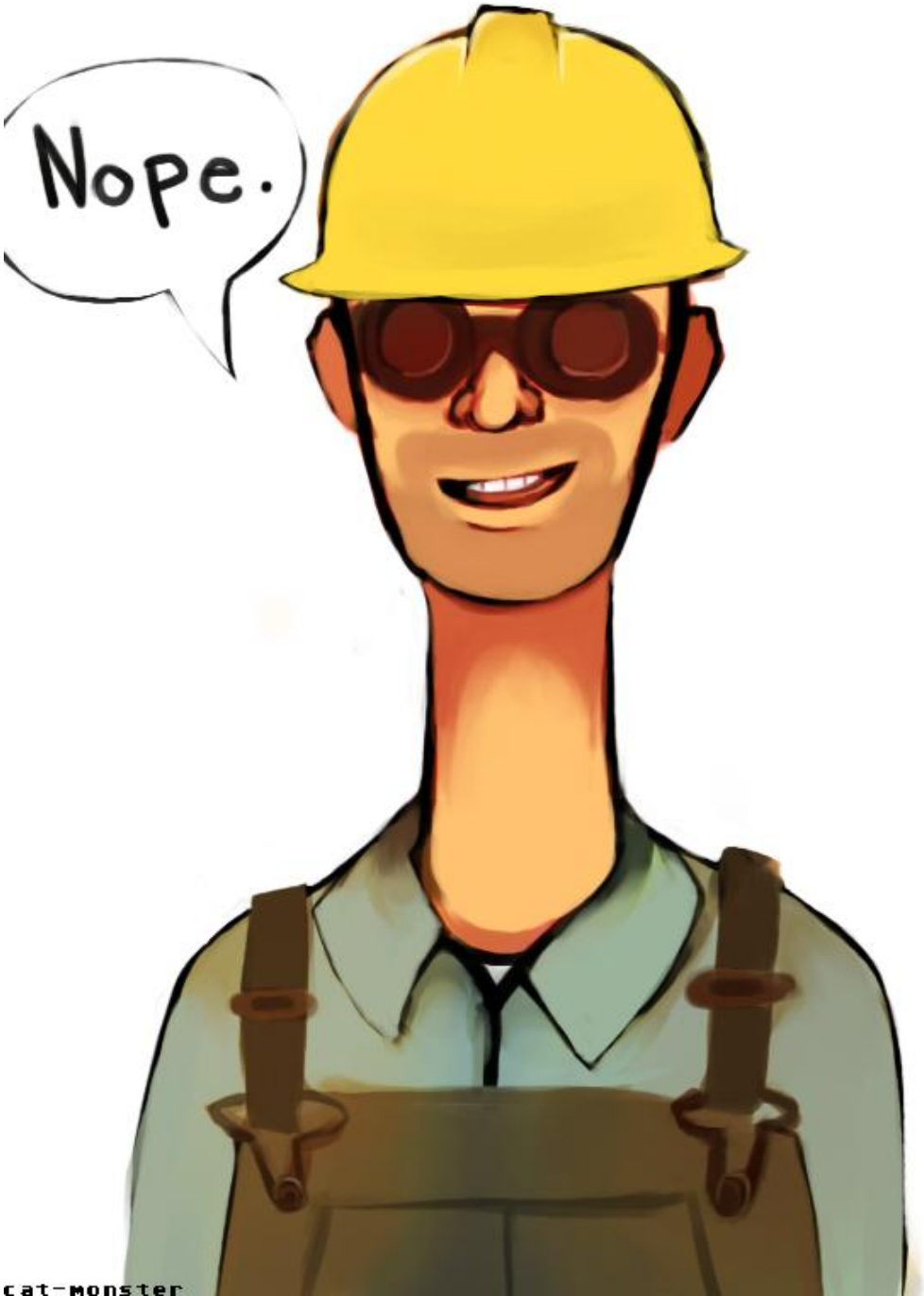
The Halting Problem

```
def myProgram(program, argument):
    if halts(program, argument):
        while(True):
            pass
    else:
        return None
```

The Halting Problem

```
def myProgram(program, argument):
    if halts(program, argument):
        while(True):
            pass
    else:
        return None
```

```
>>> myProgram(myProgram, myProgram)
```



cat-monster