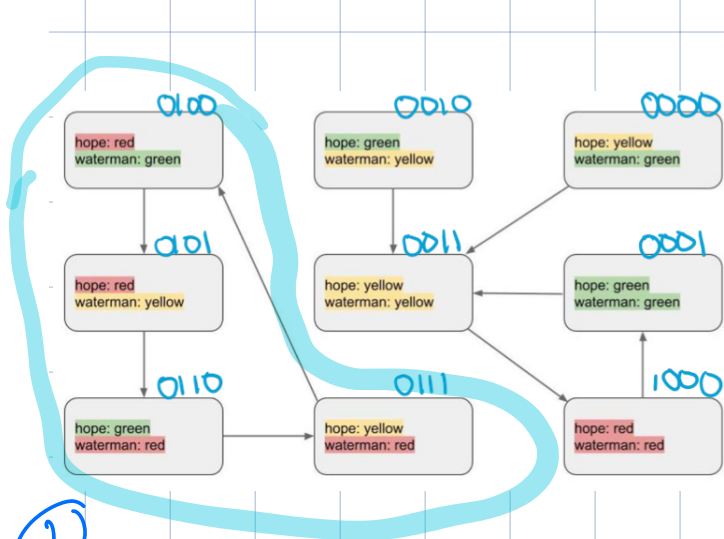


Continuing from last class: goal was to succinctly represent set of "safe" program states



①

Our goal: represent set of safe states

Could write as:  
List[0100, 0101, 0110, 0111]

This is much smaller than objects,  
but still grows linearly!

②

Idea: give each state a name

- Smallest piece of information in a computer is a bit (0 or 1)
- Assign name to each state "bit string" (sequence of bits)

For n states, how many bits?  $\log_2(N)$

2 STATES  $\Rightarrow$  REPRESENT AS BITS: 0, 1  $\Rightarrow$  1 BIT

4 STATES  $\Rightarrow$  4 POSSIBLE BITSTRINGS: 00 01 10 11  $\Rightarrow$  2 BITS

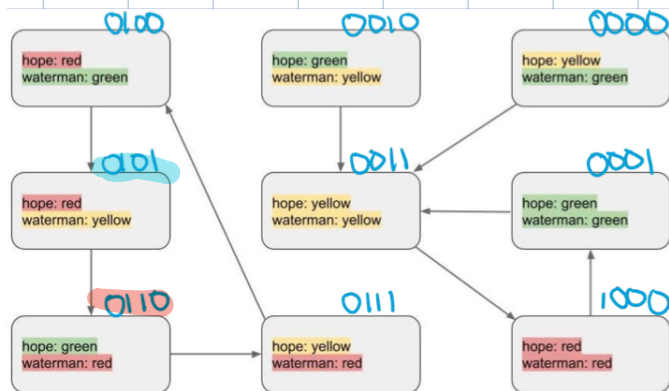
8 STATES  $\Rightarrow$  8 POSSIBLE BITSTRINGS: 000 001 010 011 100 101 110 111  $\Rightarrow$  3 BITS

N bits  $\Rightarrow 2^N$  possible "things" (in this case states)

OR

M states  $\Rightarrow \log_2(M)$  bits

$\log_2(9) = 3.1 \Rightarrow 4$  (need all strings to be same length)



=> Using this idea, we can represent each state as a bitstring of 0's and 1's

0 1 0 0  
b<sub>1</sub> b<sub>2</sub> b<sub>3</sub> b<sub>4</sub>

> Notation: we write state of individual bit as b<sub>0</sub>, b<sub>1</sub>, ..., where b<sub>0</sub> could be 0 or 1 (like a Boolean variable)

What can we do with this?

- Represent states VERY succinctly (lower space complexity)
- Can represent set of reachable states as a decision tree => Binary Decision Diagram (BDD)

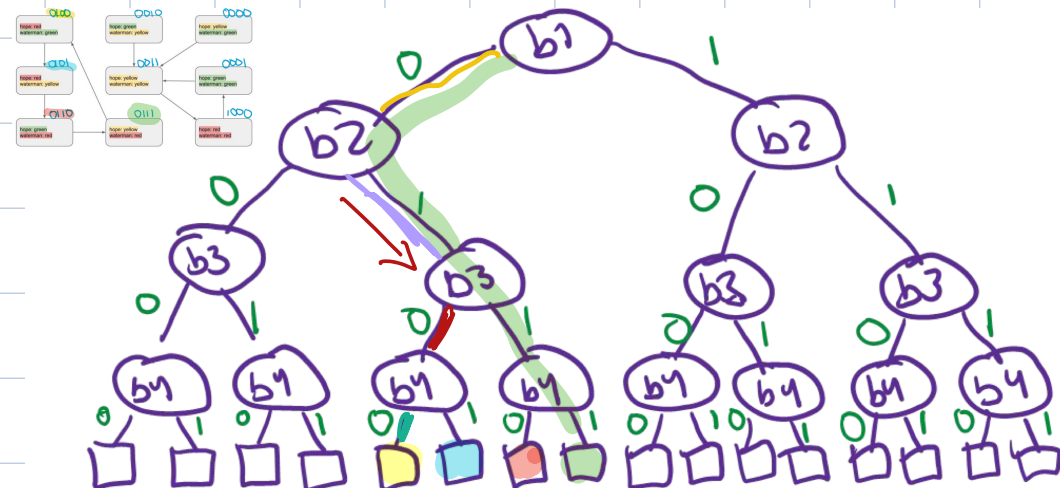
BDD: How it works (high level)

- Run BFS to find reachable states
  - Mark each leaf of BDD as reachable
- for each state
- Can make a logical formula (a boolean expression) to represent reachable states

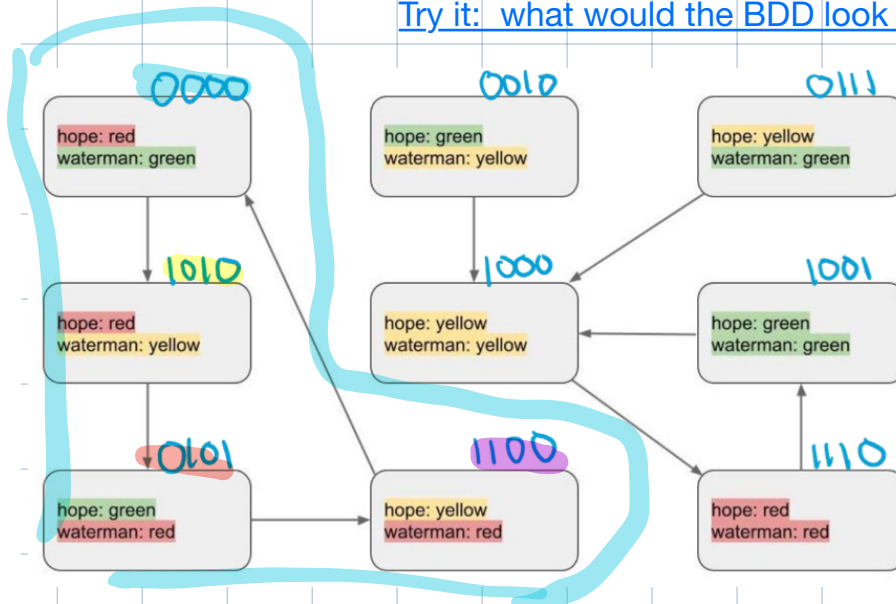
Reachable: (b<sub>1</sub> == 0) AND (b<sub>2</sub> == 1)

Could also write as: (not b<sub>1</sub>) and (b<sub>2</sub>)

=> To check if the state is safe, Don't even need to store a list of reachable states! Just need to check if the bitstring matches the formula!



Try it: what would the BDD look like if we picked these names instead?



0 0 0 0  
B<sub>1</sub> B<sub>2</sub> B<sub>3</sub> B<sub>4</sub>

Formula would be a lot more complicated!

((not b<sub>1</sub>) and (not b<sub>2</sub>) and (not b<sub>3</sub>) and (not b<sub>4</sub>)) OR  
((b<sub>1</sub>) and (not b<sub>2</sub>) and (b<sub>3</sub>) and (not b<sub>4</sub>)) OR

...

### Takeaways:

- Coming up with a good naming of states is nontrivial

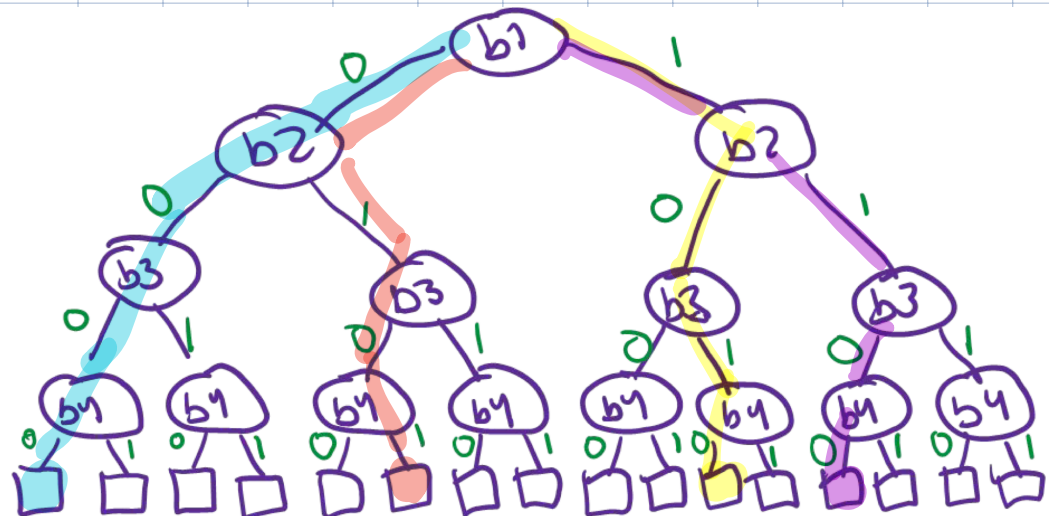
=> Problems are provably computationally hard, people work on this for specific settings

- Depending on what the BDD looks like, logical formula may get complex

Want to learn more?

Consider taking: Logic for Systems

=> Learn about algorithms people have made to represent these in more concise ways, make tools to help, software to use



Compression: how to store text efficiently?

"NATS"  
↳ 8 BITS/CHAR (ASCII)  
32 BITS/CHAR (UNICODE)

ASCII: Use 8 bit bitstring for each character  
=>  $4 * 8 \text{ bits} = 32 \text{ bits total}$

Storage: increases linearly with number of characters

BigWiki: 141 million characters \* 8 bits  $\approx$  136MB

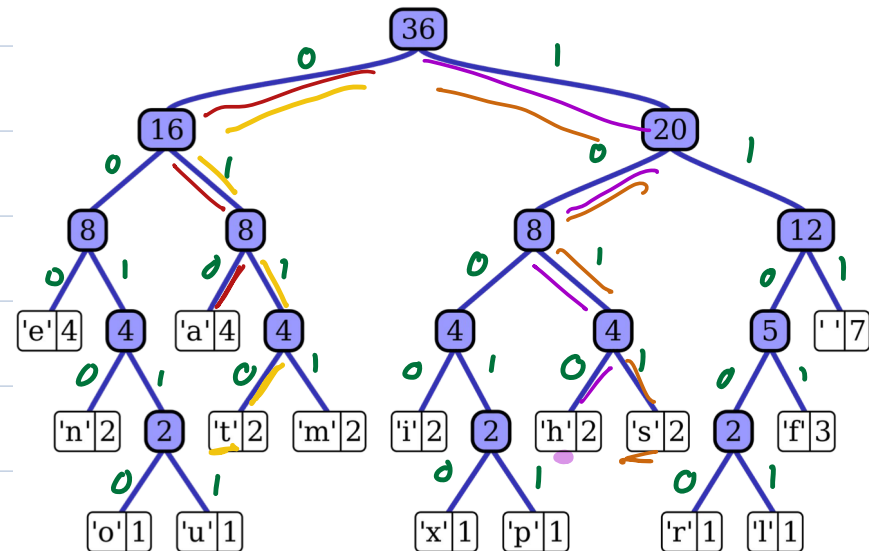
Idea: language (and most data) isn't random  
Compression: use fewer bits to represent data that appears more frequently  
"e" appears more often than "x"—so maybe we can use fewer bits?

Huffman coding:  
- for some input data, find unique bitstring to represent characters, use fewer bits for more common letters

Takeaways:

=> Leveraging patterns in data for efficiency  
=> Encoding pattern in BDD structure (use to convert)

"this is an example of a huffman tree"



[https://en.wikipedia.org/wiki/Huffman\\_coding](https://en.wikipedia.org/wiki/Huffman_coding)

"NATS"  
h: 1010 4  
a: 010 3  
t: 0110 4  
s: 1011 4

=> 15 BITS

a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

