

Puzzle

(State Transition Diagram)

Rupesh Nasre.
rupesh@cse.iitm.ac.in

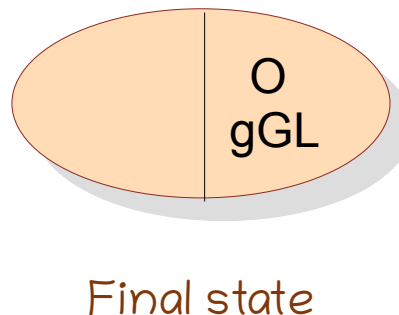
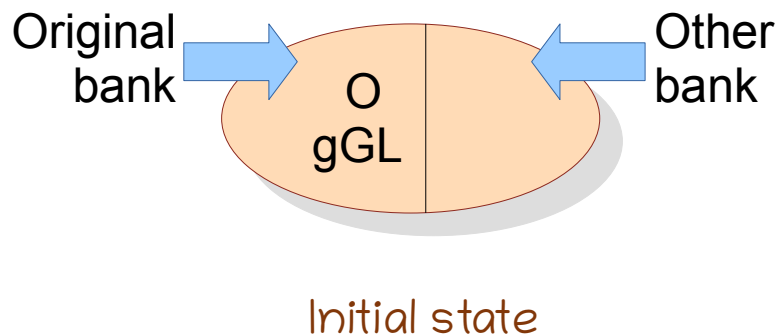
Grass, Goat, Lion Puzzle



- Owner with grass, goat, lion at a river bank
- Boat with capacity 2
- All of them need to travel to the other bank.
- All the entities are well-behaved in owner's presence.
- In the absence, goat will eat the grass; lion will eat the goat.
- Lion won't eat the grass.

A Solution

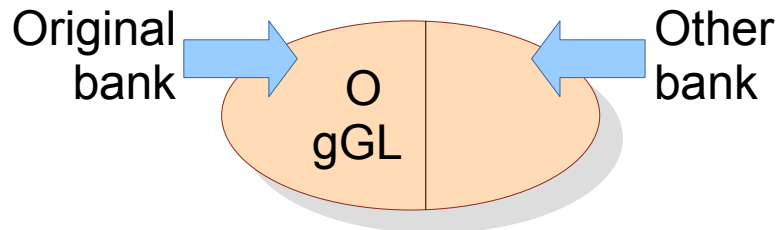
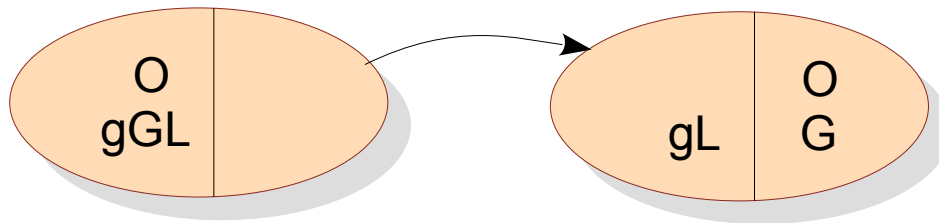
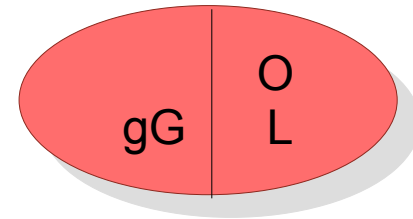
- From youtube
- Let's model it using **states**.
- A **state** denotes a situation in the solution.
 - Not all the states are relevant.
 - Only those where there is change are relevant.
 - What is the initial state? Final expected state?



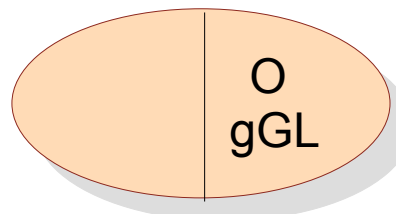
A state will correspond to our data structure.

States and Transitions

- Some states are unsafe.
 - Specific to this problem
- Boat movement changes the state. This is depicted with a **transition**.



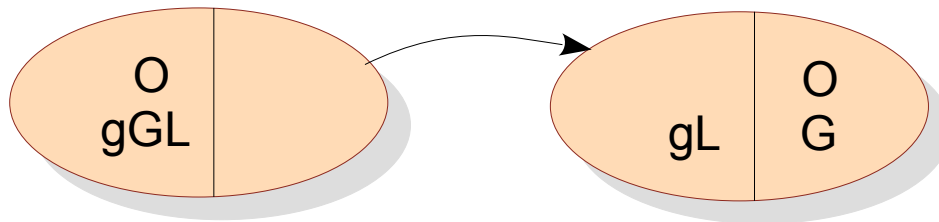
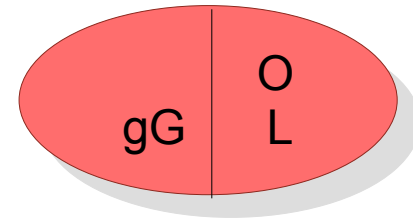
Initial state



Final state

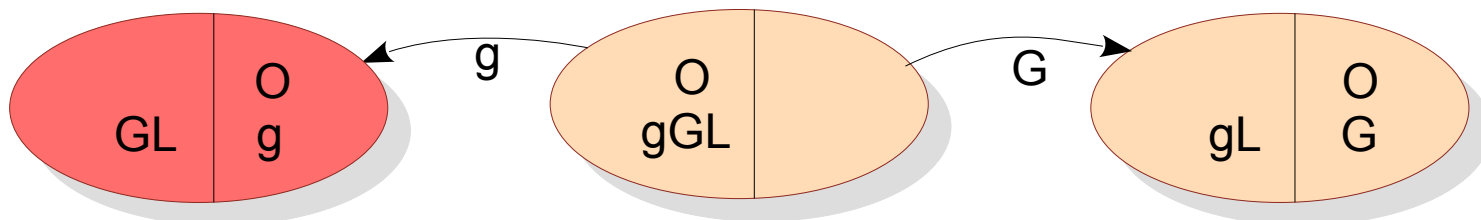
States and Transitions

- Some states are unsafe.
 - Specific to this problem
- Boat movement changes the state. This is depicted with a **transition**.



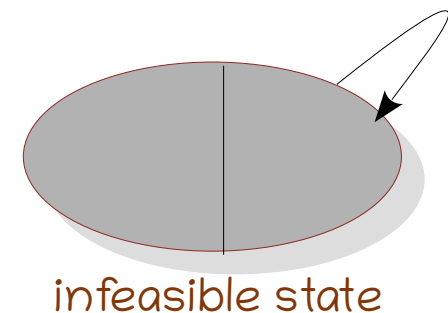
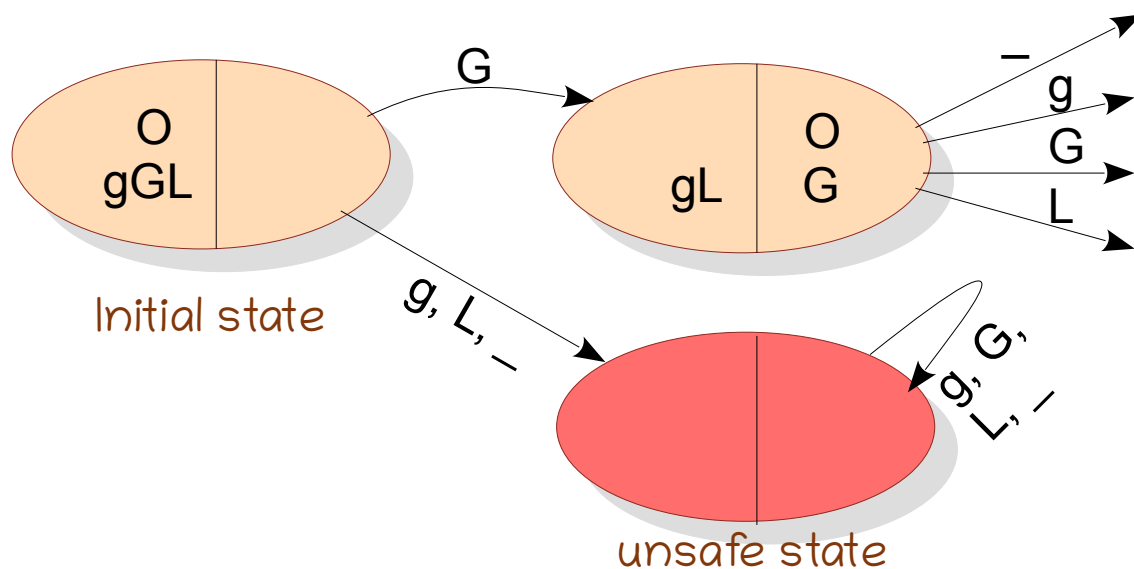
- Multiple transitions are possible out of a state. So we label a transition.

How many more transitions are possible out of the middle state?

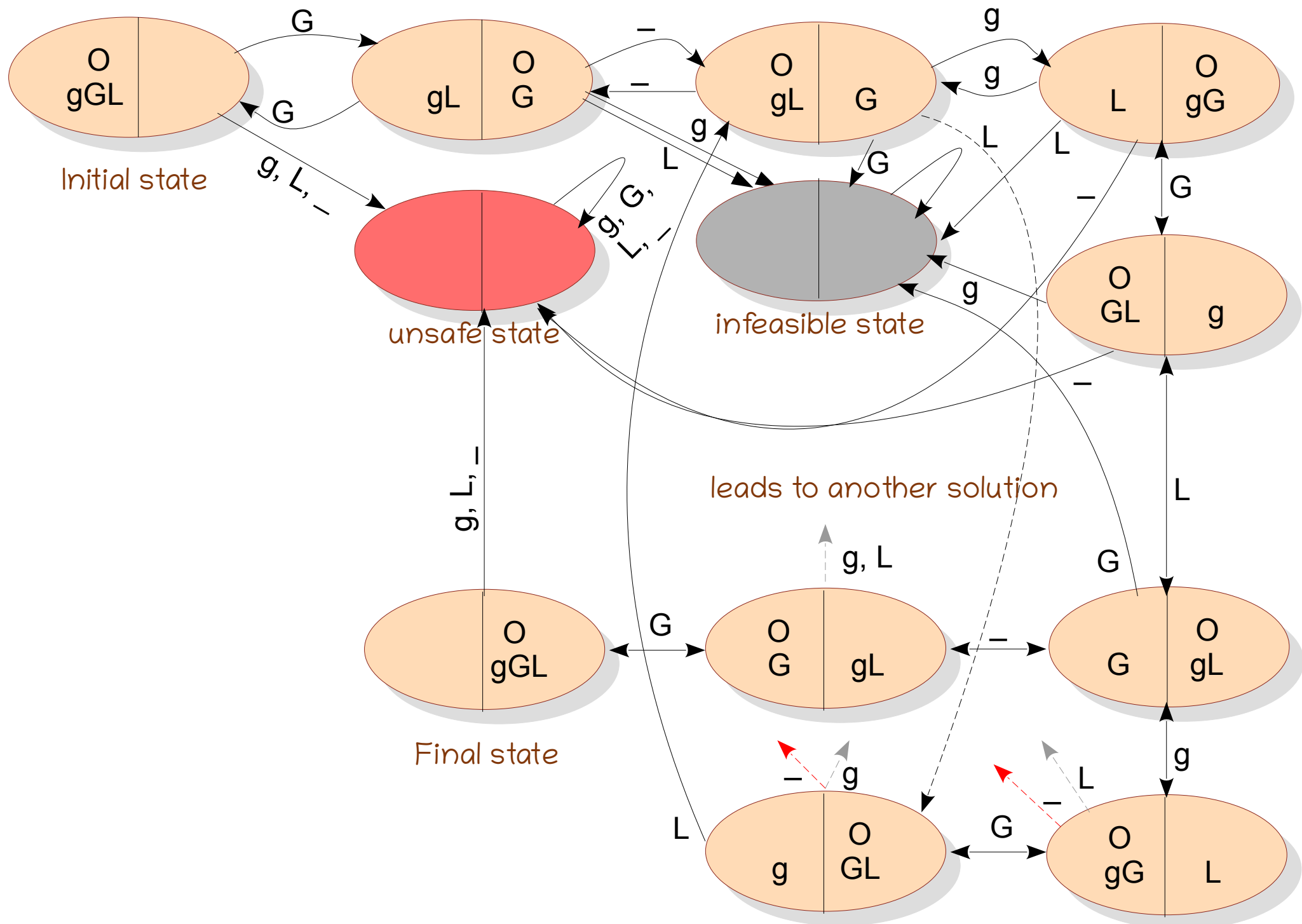


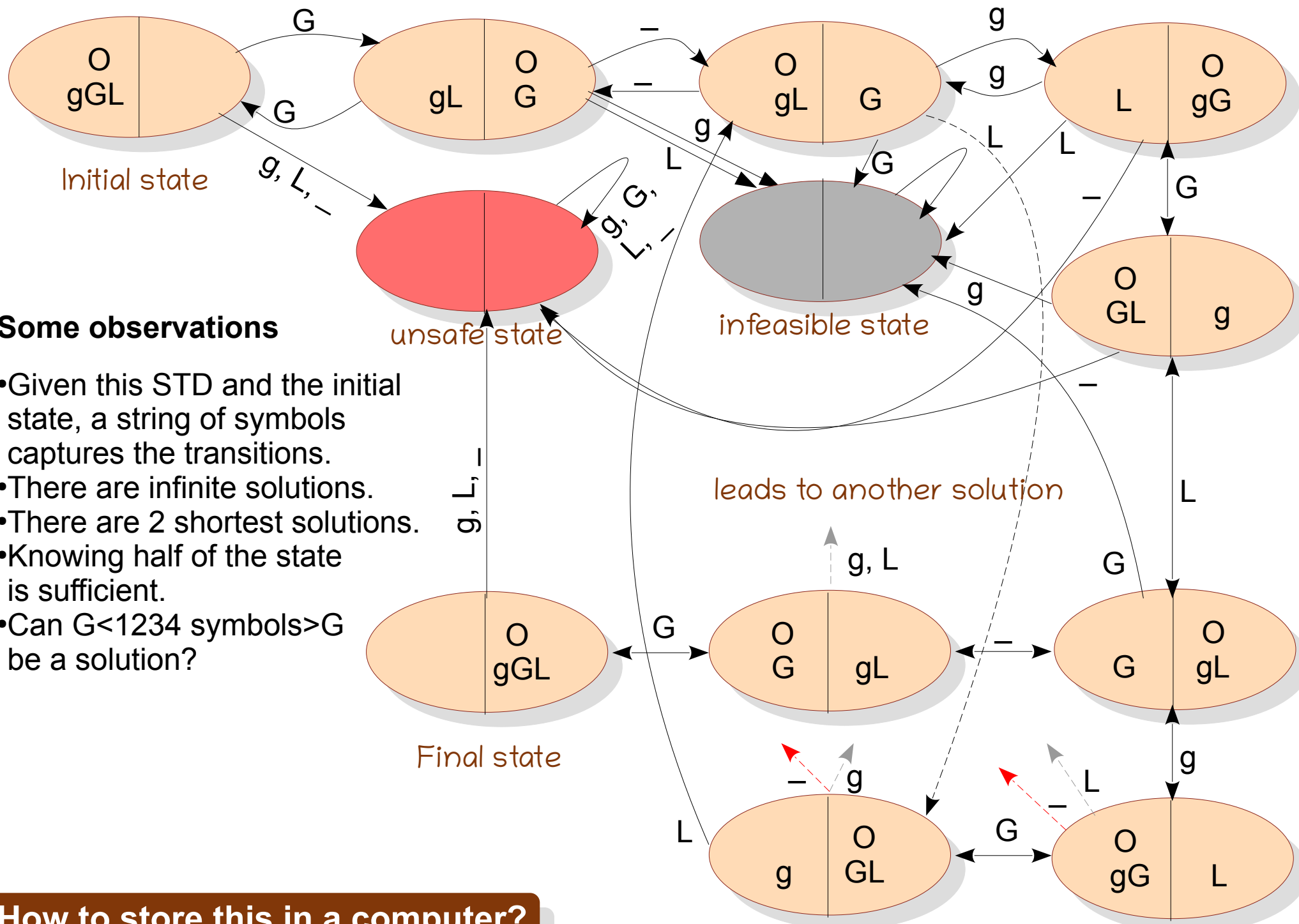
Solution

1. Build the state transition diagram. Reuse states.
 2. Find if a path exists from the initial state to the final state, without going through an unsafe state.
- Alternatively, an unsafe state may remove an entity.
 - Alternatively, an unsafe state may not have a transition outside.



Classwork: Draw the full diagram.





Some observations

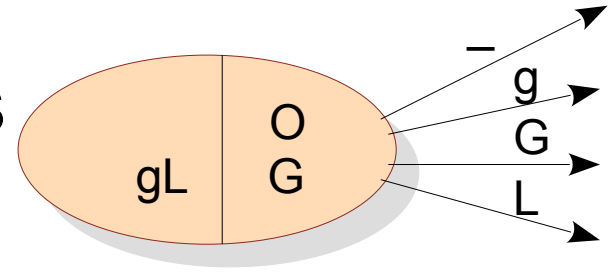
- Given this STD and the initial state, a string of symbols captures the transitions.
- There are infinite solutions.
- There are 2 shortest solutions.
- Knowing half of the state is sufficient.
- Can $G<1234\text{ symbols}>G$ be a solution?

How to store this in a computer?

Storage in a Computer

- A (half)state can be modeled as

- four boolean flags
- four transition pointers



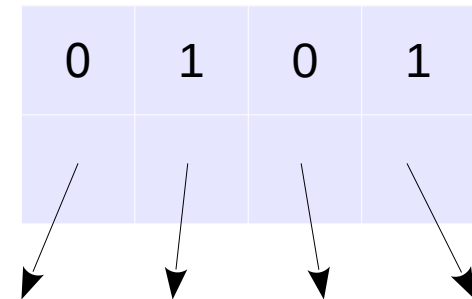
- An STD can be modeled as

- a pointer to the initial state

```
struct State {  
    int O, g, G, L;  
    struct State *p_  
        , *pg,  
        *pG, *pL;  
    // int visited;  
};
```

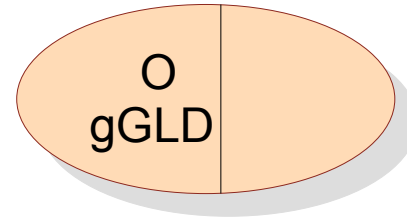
- To find a solution

- we need the final state
- we need to avoid infinite loops



**How would your code change for the following generalization?
Add a Dinosaur D who eats a Lion (and does not eat a Goat or grass).**

Puzzle Generalization



- gGLD
- Draw transitions from the initial state.
 - All would lead to the unsafe state.
- We need a larger-capacity boat.
 - How much capacity?
 - The transition was earlier a symbol, now it is a string.
 - Whom will the owner carry? What are the unsafe situations?
 - Unsafe situations must be accompanied by the owner.

How does the solution look like?

Puzzle Generalization

- $gGLD \mid \rightarrow gL \mid GD \rightarrow \mid gGLD$
- $gGLD \mid \rightarrow gD \mid GL \rightarrow gGD \mid L \rightarrow G \mid gLD \rightarrow$
 $GL \mid gD \rightarrow \mid gGLD$

Once syntax is defined, we can represent STD with a textual specification.

How about further generalizing it to 5 entities?

Puzzle Generalization

- abcde |
 - How much boat capacity is needed?
 - How do we solve it?
- abcde | $\rightarrow$ ace | bd $\rightarrow$ e | abcd $\rightarrow$ bde | ac $\rightarrow$ bd | ace $\rightarrow$ | abcde
- How does abc with 1 capacity look like?
 - abc | $\rightarrow$ ac | b $\rightarrow$ c | ab $\rightarrow$ bc | a $\rightarrow$ b | ac $\rightarrow$ | abc
- For even no. of entities, take odd, then take even.
- For odd no. of entities, take even, come back, take all odd but one, bring back all even, take the remaining odd, come back, take all even.

Homework: Other Generalizations

- Eating cycle: first entity can eat the last one
 - $a \leftarrow b \leftarrow c \leftarrow d \leftarrow e \leftarrow a$
 - Rock-Paper-Scissors
- Duplicate entities
 - 1 grass, 2 goats, 1 lion
- Subsumed eating
 - a can be eaten by bcde, b can be eaten by cde, ...
- Arbitrary eating structure
 - d can eat a and b but not c, b does not eat a, e can eat a and b, a can eat d, ...

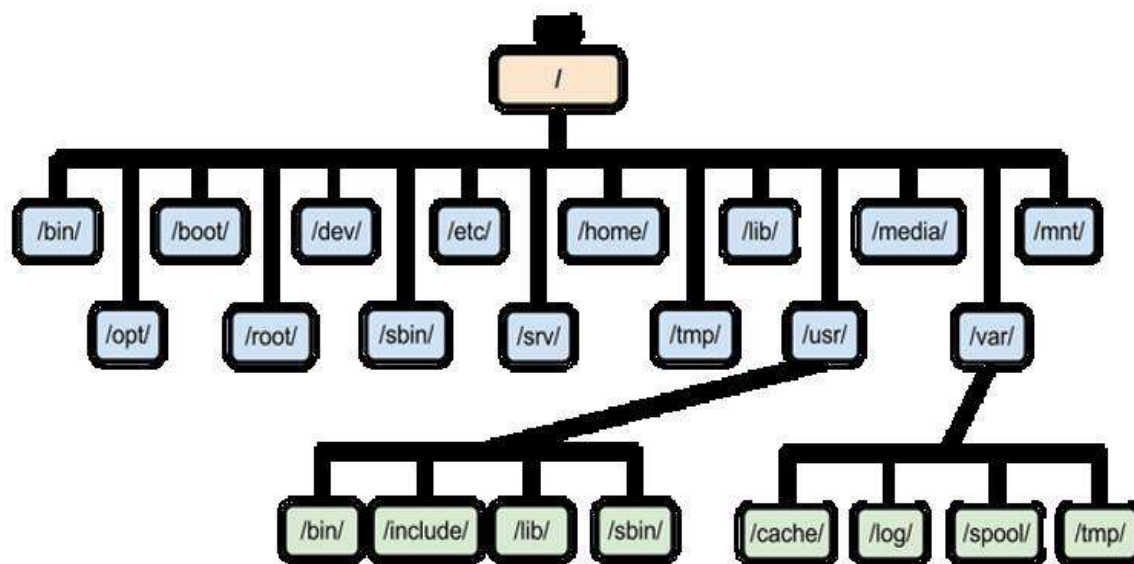
Terminal and Editor

- Text-only mode, mouse not useful
- Ctrl-Alt-F3 / F4 / ... from the graphics mode
- Ctrl-Alt-F1 / F7 / ... to go back to the GUI.
- We cannot run many graphics applications
 - such as chrome, openoffice, vscode, ...
- We can edit .c files, compile and execute them.
 - emacs hello.c
 - vi hello.c
 - gcc hello.c
 - ./a.out

First step to being liberated is to feel handicapped.

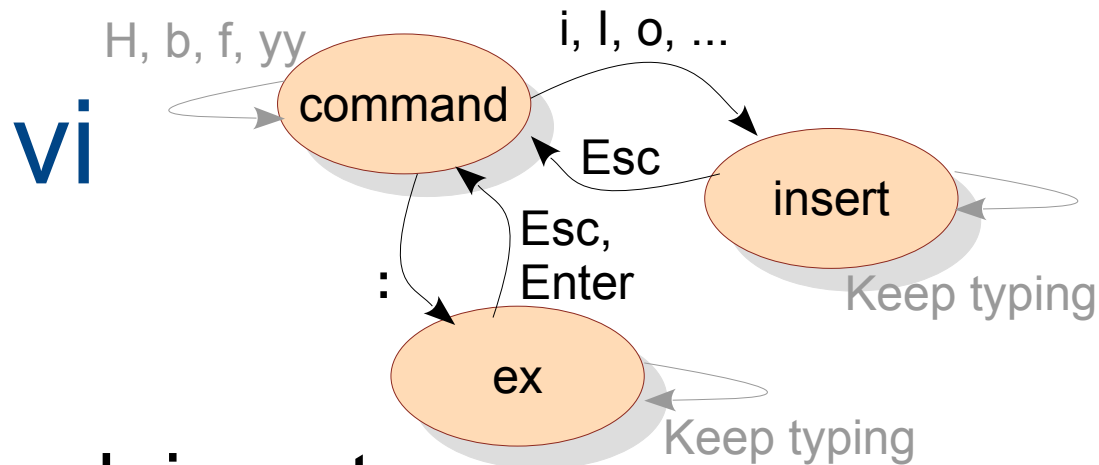
Terminal Commands

- Readonly: ls, pwd, cat, cd
- Create: cp, mkdir, vi, emacs
- Remove: rm, rmdir, mv
- Others we will learn later (sed, grep, awk, ...)
- Basic terminal commands



emacs

- Direct editing (unlike vi)
- Uses lisp interpreter (more in CS3100)
- Start with: `emacs filename`
- Exit with: `Ctrl-x Ctrl-c`
- [Basic commands, cheatsheet](#)
- If you wish to be an expert in emacs: force yourself to learn and use its commands.

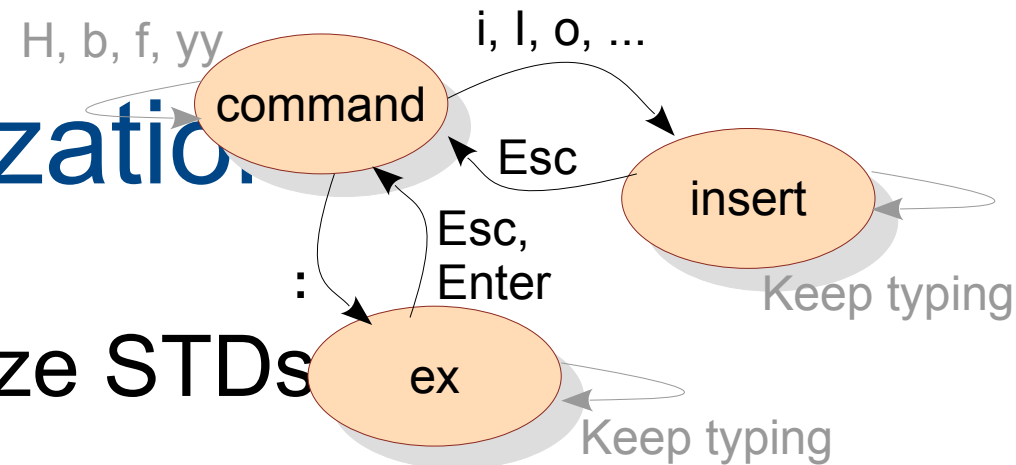


- `vi` visual editor
- Three modes: `command`, `insert`, `ex`
 - Command: editing actions
 - Insert: Typing
 - Ex: Additional commands (e.g., setting params)
- Start with: `vi filename`
- Exit with: `<Esc>:wq<Enter>`
- Basic commands, cheatsheet
- If you wish to be an expert in `vi`: force yourself to learn and use its commands.

Classwork with vi and emacs

- Create a file message.txt containing 5 lines.
- Delete the first line.
- Move third line to the end.
- Save the file.
- Copy the second word of the fourth line on a new line at the end.
- Copy the file to <rollnumber>.txt.
- Delete message.txt
- Rename the file back to message.txt.
- Move the file to a new directory L1.

Visualization



- We would like to visualize STDs
- Tool called graphviz (webtool called [webgraphviz](#)), CLI tools such as dotty
- Uses a .dot format.

```
graph g {  
    a -- b  
}
```

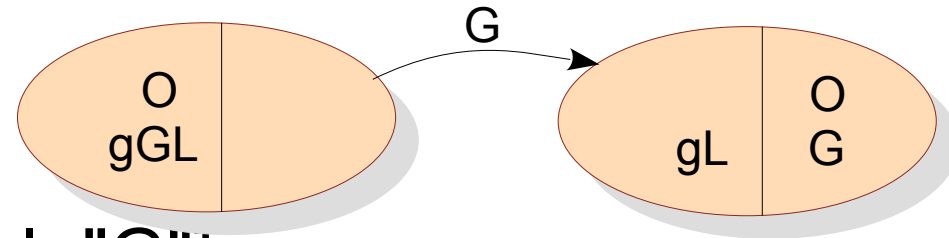
```
graph g {  
    a -- b  
    b -- c -- a  
}
```

```
digraph g {  
    a -> b  
    b -> c -> a  
}
```

```
digraph vi {  
    command -> insert [label="i, l, o, ..."]  
    insert -> command [label="Esc"]  
    insert -> insert [label="Keep typing"]  
    command -> ex [label=":"]  
    ex -> command [label="Esc, Enter"]  
    command -> command [label="H, b, f, yy, ..."]  
    ex -> ex [label="Keep typing"]  
}
```

Visualization

- Coming back to our STD



- OgGL_ -> gL_OG [label="G"]
- Overall

```
digraph STD {  
  unsafe [color=red, style=filled]  
  infeasible [color=gray, style=filled]  
  
  OgGL_ -> gL_OG [label = "G"]  
  OgGL_ -> unsafe [label = "gL_O"]  
  
  gL_OG -> OgGL_ [label = "G"]  
  gL_OG -> infeasible [label = "gL"]  
  gL_OG -> OgL_G [label = "O"]  
  
  ...  
}
```

In this course...

Application	Programming	Tools
Puzzle of grass, goat, lion	Pointers, structures	Editor, visualization
Data maintenance	File I/O	I/O redirection, make
Item grouping	Linked Lists	Debugging
Gene sequencing	Recursion, strings	Memory profiling, ulimit
Closest match	Matrices	Compute profiling, plotting
Keep the recent	Arrays	Version control
Cryptography	Strings, bitwise operators	Library (ncurses)
Unit testing	Command-line	Testing
Web-page creation, analysis	HTML, Strings	wget, shell scripting
Game	Graphics	Library (graphics)
Basics	Python	--

Our goal: *to have fun with automation*