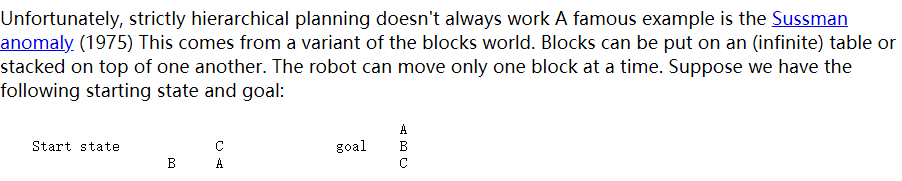
ECE 448 Midterm1 possible questions

**Shakey and Blocks world：** (1966-72) the world's first mobile "automaton." A∗ search was developed around 1968 as part of the Shakey project at Stanford.

Towards the end of the project, it was using a PDP-10 with about 800K bytes of RAM. The programs occupied 1.35M of disk space. This kind of memory starvation was typical of the time. For example, the guidance computer for Apollo 11 (first moon landing) in 1969 had only 70K bytes of storage.



So strictly hierarchical planning doesn't work. We need a more flexible approach to planning. For example, expand what's required to meet **subgoals**, then try to order all the little tasks. So you can "interleave" sub-tasks from more than one main goal.

**Waltz line labelling** Constraint propagation was developed by David Waltz in the early 1970's for the Shakey project. The original task was line labelling (from Lana Lazebnik based on David Waltz's 1972 thesis).

**STRIPS planning：** STRIPS planning, or the Stanford Research Institute Problem Solver, was developed in 1971. It is a simple version of classical planning, which is used to manage the high level goals and constraints of a complex planning problem, like waypoints for a robot doing assembly/disassembly tasks.

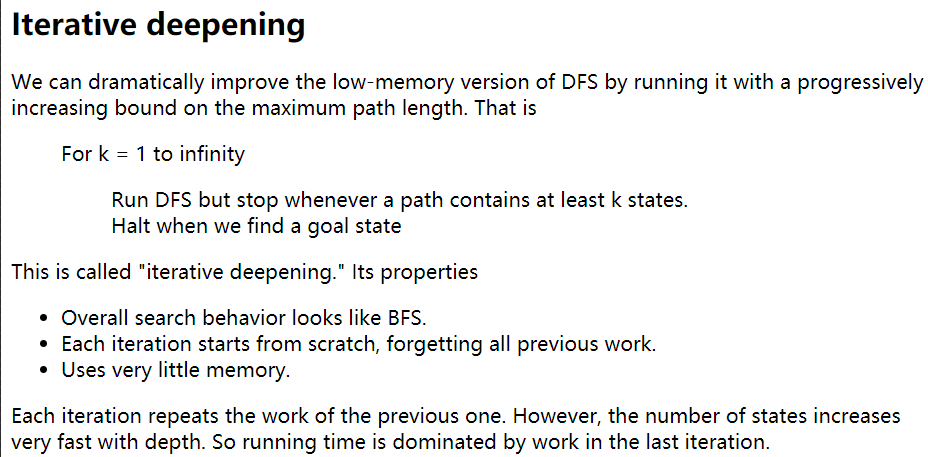
**Boston Dynamics**:

We often imagine intelligent agents that have a sophisticated physical presence, like a human or animal.

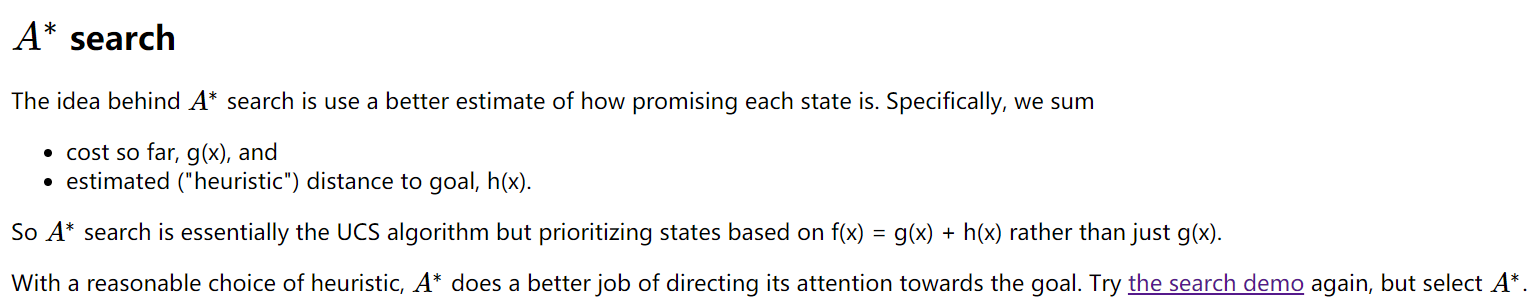
robot falling down 2017

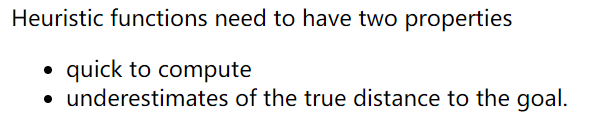
**Google self-driving bike** fake

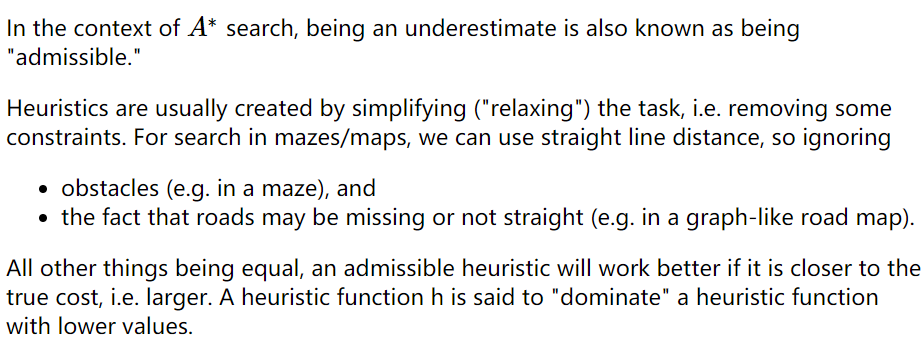
**McCulloch and Pitts**: early neural nets, 1940s

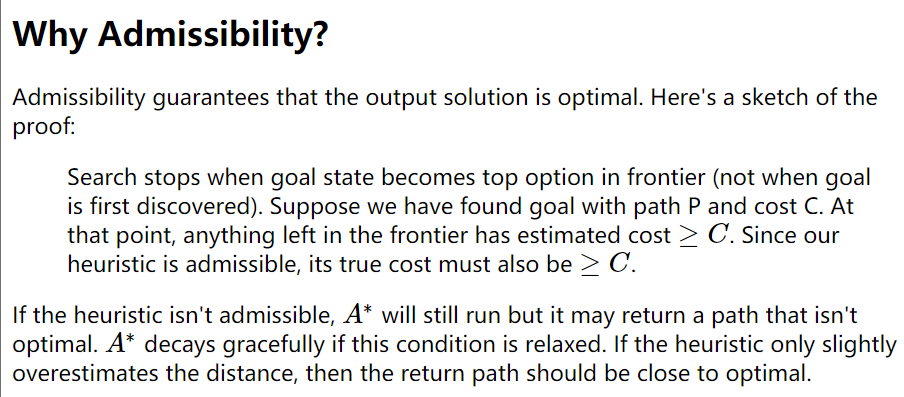


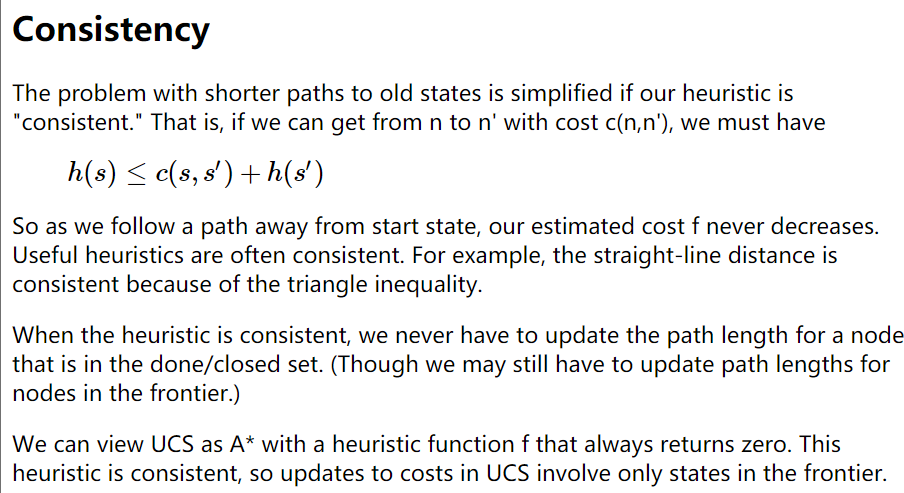
The main change between BFS and UCS is that the frontier is stored as priority queue. In each iteration of our loop, we explore neighbors of the best state (the shortest path length) seen so far.

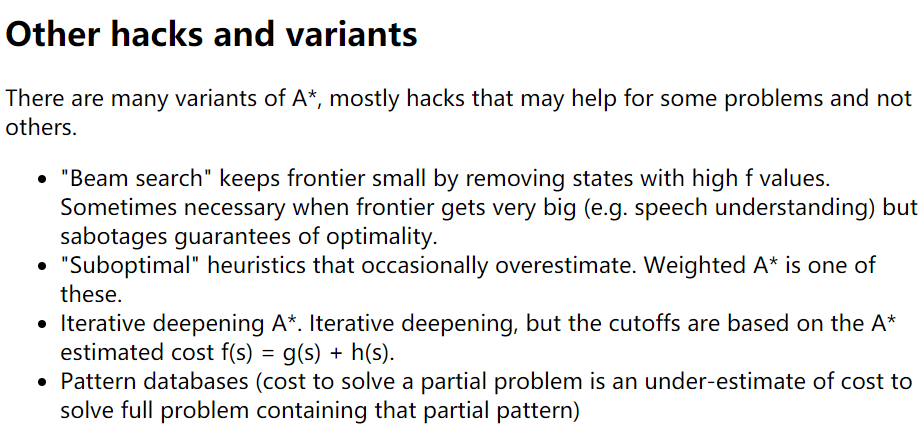


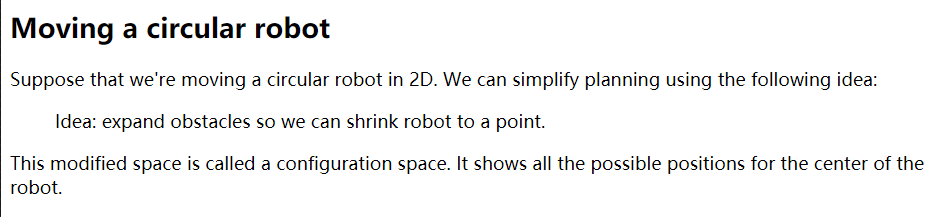


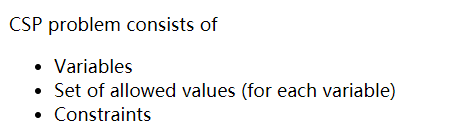




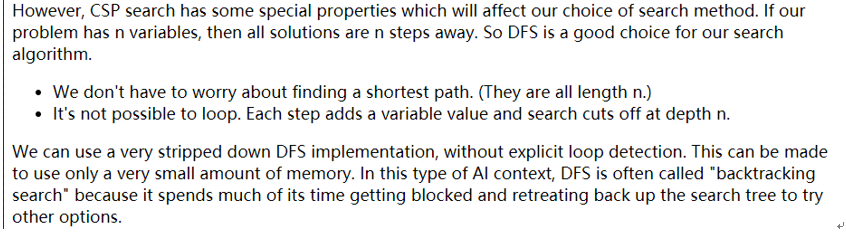






 **Constraint Satisfaction Problems**

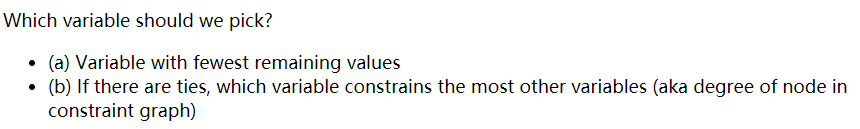
**Notice that graph coloring is NP complete.** We don't know for sure if NP problems require polynomial or exponential time, but we suspect they require exponential time. However, many practical applications can exploit domain-specific heuristics (e.g. linear scan for register allocation) or loopholes (e.g. ok to have small conflicts in final exams) to produce fast approximate algorithms.



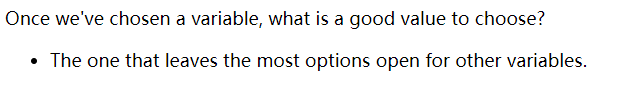
When to stop and backtrack?

Smart method (**forward checking**): During search, each variable keeps a list of its possible values. At each search step, remove values from these lists if they violate constraints, given the values we've already assigned to other variables. Back up if any variable has no possible values left.

Heuristics for variable assignments

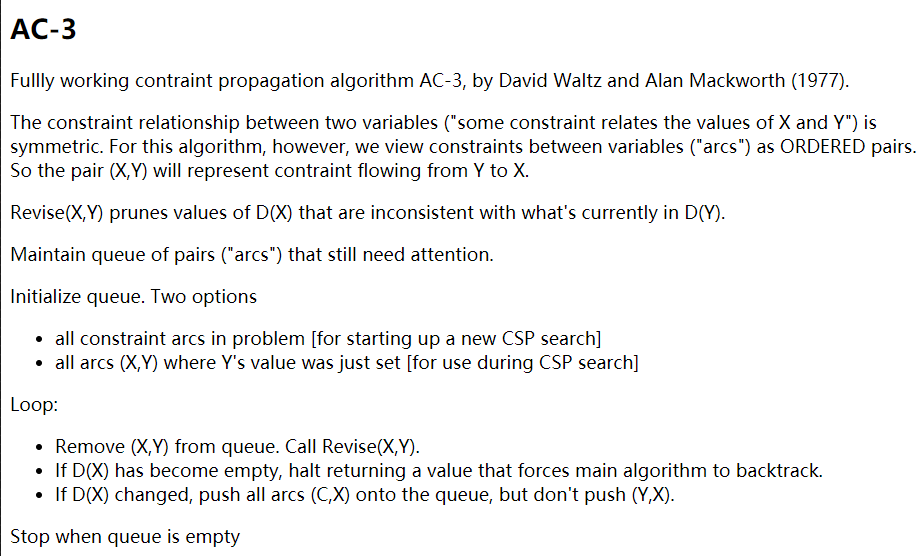


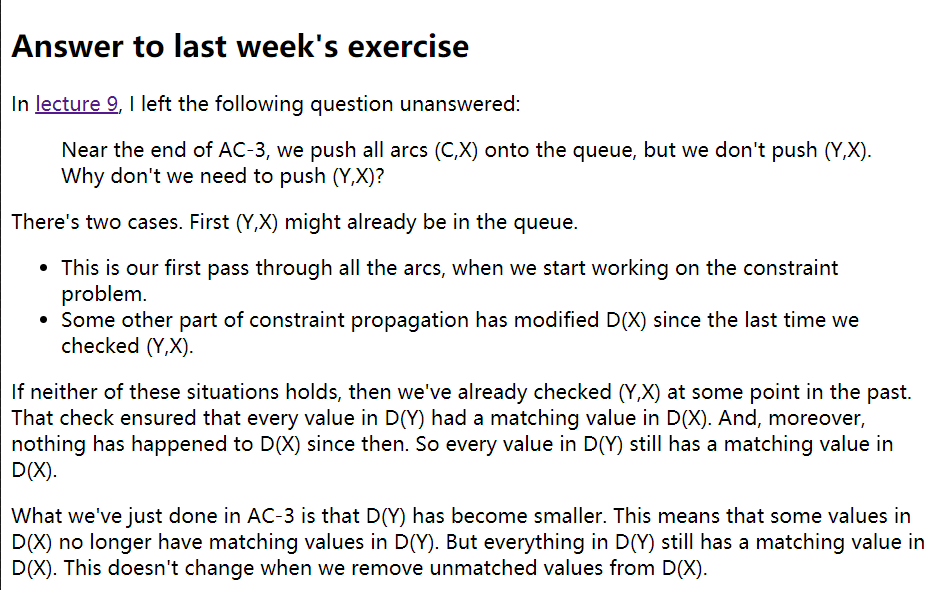
Choosing a value



**Constraint propagation**

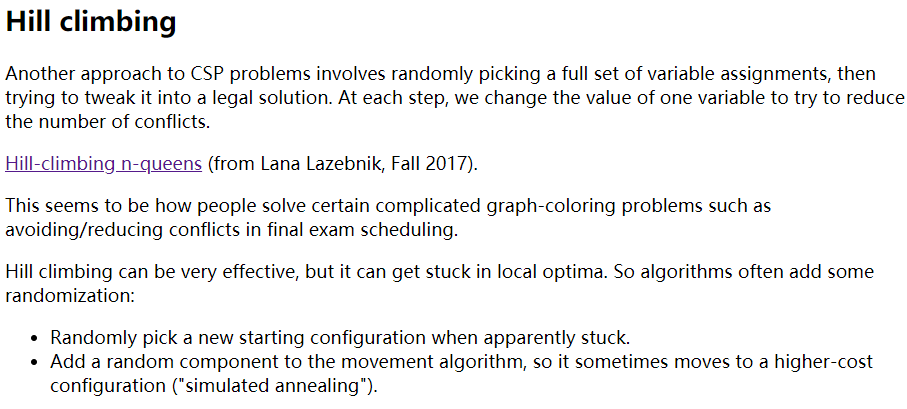
When we assign a value to variable X, forward checking only checks variables that share a constraint with X, i.e. are adjacent in the constraint graph. This is helpful, but we can do more to exploit the constraints. Constraint propagation works its way outwards from X's neighbors to their neighbors, continuing until it runs out of nodes that need to be updated.



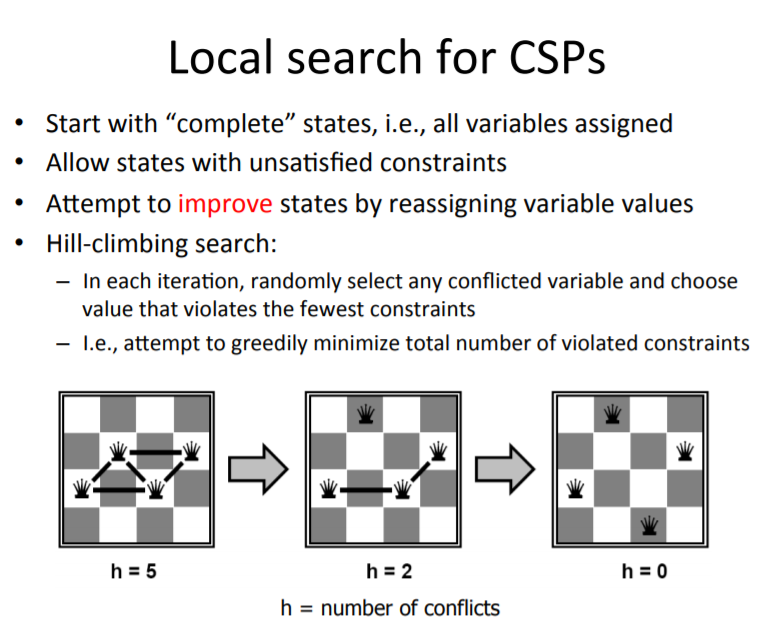


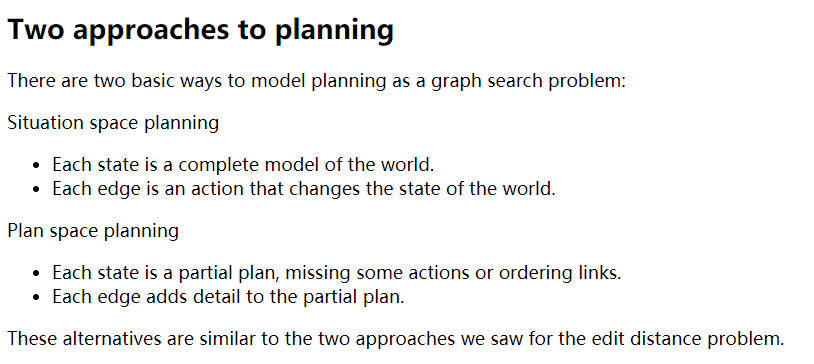
Briefly: why not push (Y,X):

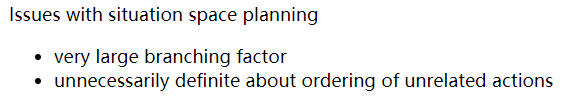
Because it has already been checked or already in the queue.

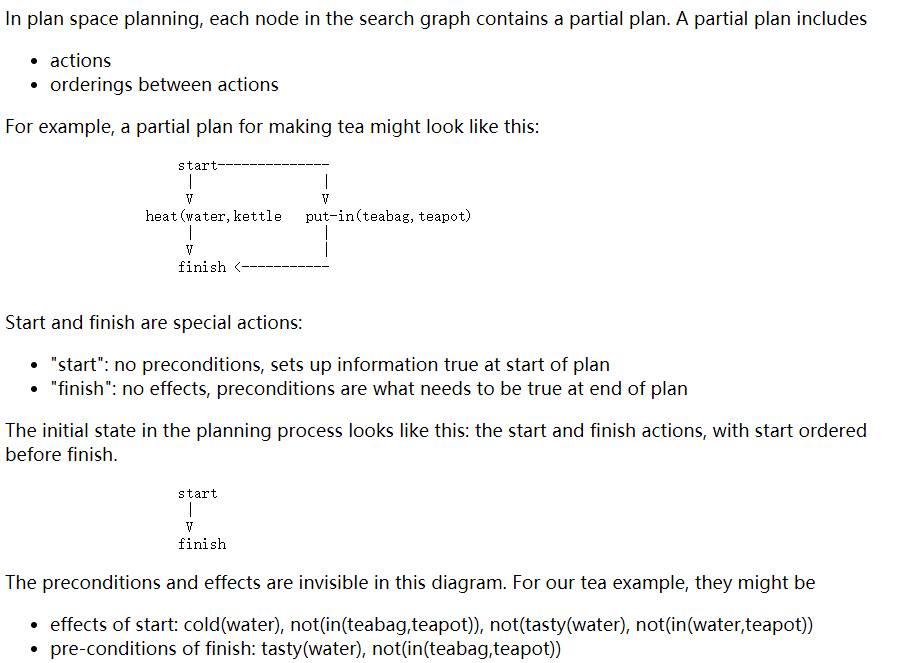


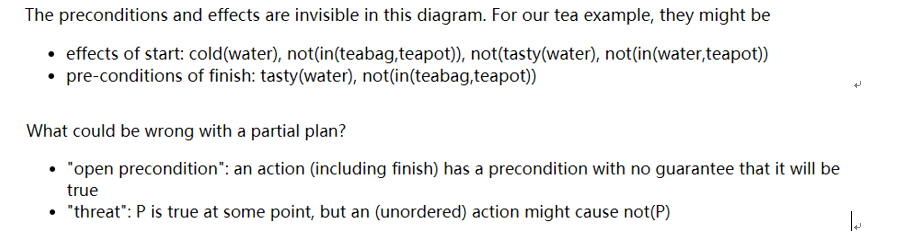
**Hill climbing can be very effective, but it can get stuck in local optima.**

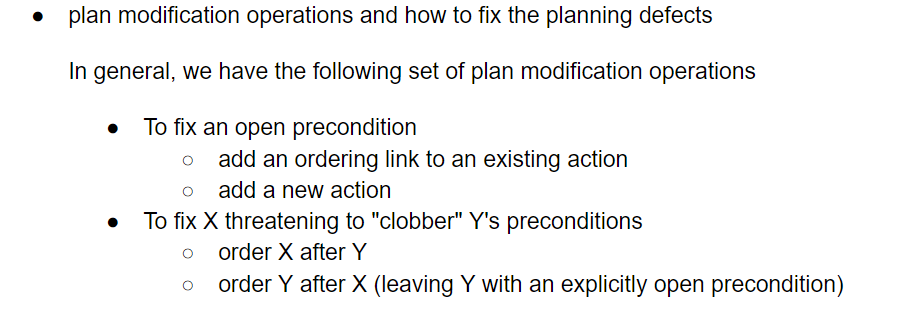


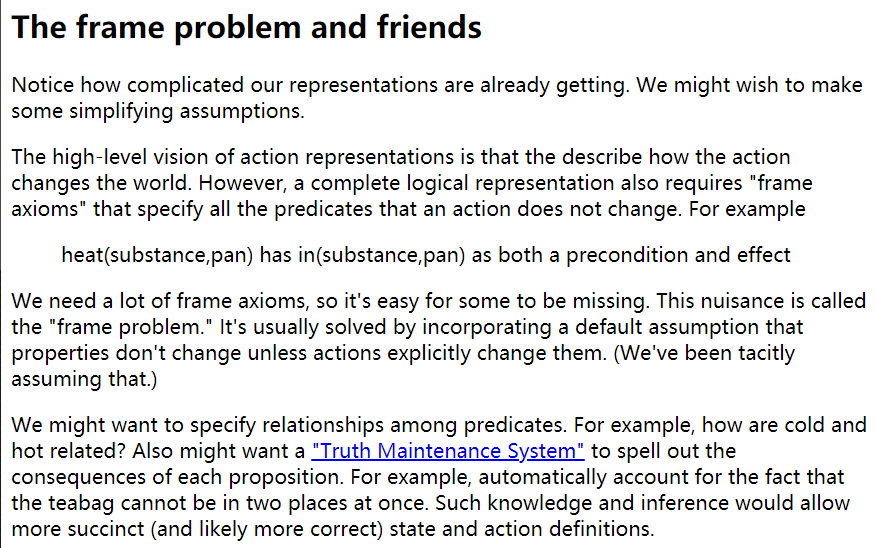




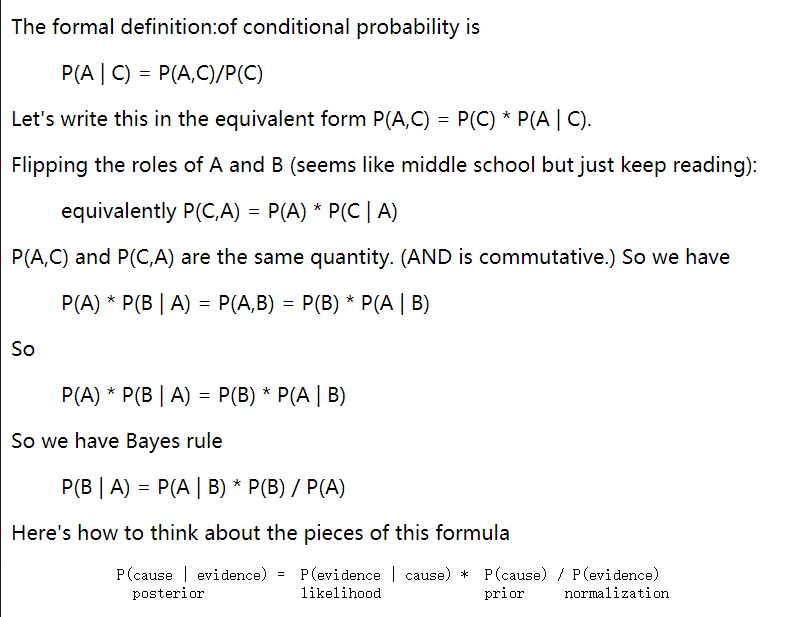








The frame problem is saying that we need lots of frame axioms so that we can properly account for all of the things that are changed and not changed by actions. It is easy to make the mistake of not accounting all frame axioms.  And so, instead, we should assume by default that properties don't change unless the actions explicitly change them.



P(evidence | cause) tends to be stable, because it is due to some underlying mechanism.

P(cause | evidence) is less stable, because it depends on the set of possible causes and how common they are right now.

