Ch1 Informed Search

1. State space graph G(V, E) < V: state ; E: arcs: sucessors with length = cost
2. Search tree：依次往下 expand sucessor as child
(1) maintain a partial plan as expanding

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

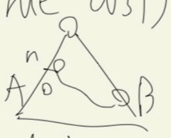
fringe, expansion
exploration

(2) DFS / BFS
① fringe: LIFO stack / FIFO queue → s: shallowest solution
② time complexity: $O(b^m)$ / $O(b^s)$ / space complexity = $O(bm)$ / $O(b^s)$
(3) Uniform Cost Search (UCS) ② if solution cost C^*, single arc at least cost ε ⇒ effective depth → C^*/ε
① 選最少cost的擴展fringe
→ g(n) (cost from root to n) → time complexity $O(b^{C^*/\varepsilon})$ = space complexity

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
   closed ← an empty set
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST(problem, STATE[node]) then return node
      if STATE[node] is not in closed then
         add STATE[node] to closed
         for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
         end
   end
```

Ch2 Uninformed Search

1. Search heuristics: a function that estimates how close to goal
2. Greedy Search: (1) expand the node (you think) nearest to goal
might lead to some worst case (最少 h(n))
3. A^* search: (1) expand by 最少 $f(n) = g(n) + h(n)$
(2) implementation: should stop when dequeue the goal
(3) what might went wrong: estimated heuristics 可以錯誤悲觀
4. Heuristics (1) admissible (optimistic): 估計heu ≤ actual costs
⇒ $0 \le h(n) \le h^*(n)$ (true cost)
optimality: $f(n) \le f(A) < f(B)$



$\begin{cases} f(n) \le f(A) \\ f(A) \le f(B) \\ n \text{ before } B \end{cases}$

(2) characteristics: ① dominance $h_a \ge h_c$ if $\forall n: h_a(n) \ge h_c(n)$
exact - max(ha, hb) ← 可以 construct semi-lattice
ha hb zero ...
5. Graph Search: (0) idea: never expand a node twice
(1) ⇒ tree search + set of expanded states (closed set)
(2) 為避免誤判是 same set (node 看似同但 path diff.) ⇒ 針對 heuristics 修正
(3) consistency: arc heuristic cost (兩node h相減) ≤ actual cost

⇒ $h(A) - h(C) \le cost(A \text{ to } C)$
(consistency ⇒ f along a path never decrease

Ch3 Constraint Satisfaction problem

1. Formulation: state X_i with values from domain D_i,
goal test by satisfying all constraint
2. Binary csp graph: nodes = var ; arc: constraints
3. Backtracking search: < one variable at one time + DFS ; check constraint as you go

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
         add {var = value} to assignment
         result ← RECURSIVE-BACKTRACKING(assignment, csp)
         if result ≠ failure then return result
         remove {var = value} from assignment
   return failure
```

檢查

4. Filtering
(1) forward checking
當assign時同時確認其他相關的var值可能值
(2) constraint propagation: ① X→Y is consistent: 對於每個箭頭出發端的every x 都應在被指端有y
② check all arc consistent
③ if X loses a value ⇒ neighbor of X should recheck 對應
△ 仍有可能 no sol.

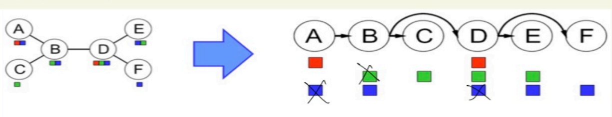
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp
   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

$O(n^2 d)$

```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi]; removed ← true
   return removed
```

$O(d^2)$

△ k-consistency:
從 1-consistency: ∀ node 都有可 assign
⇒ k-consistency: ∀任意k-1 node ⇒ k^{th} node 有consistent assign
5. Ordering: (1) minimum remaining values: 選最少可能性的 node (fail fast)
(2) least constraining value: 最不被限制
6. Structure (1) 拆解: $O(d^n) \to O(\frac{n}{c} d^c)$
7. Tree Structured CSP: $O(nd^2)$ can solve
(1) alg.



- Remove backward: For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- Assign forward: For i = 1 : n, assign X_i consistently with Parent(X_i)

(2) nearly-tree structure: ① 挑1, assign, 修其他 constraint, 拔掉
② cutset: 把①擴大至挑1 set ... ⇒ $O(d^c (n-c) d^2)$
③ tree decomposition: 用 mega node 來表示
8. Iterative algorithms

Algorithm: While not solved,
- Variable selection: randomly select any conflicted variable
- Value selection: min-conflicts heuristic:
  - Choose a value that violates the fewest constraints
  - I.e., hill climb with h(x) = total number of violated constraints

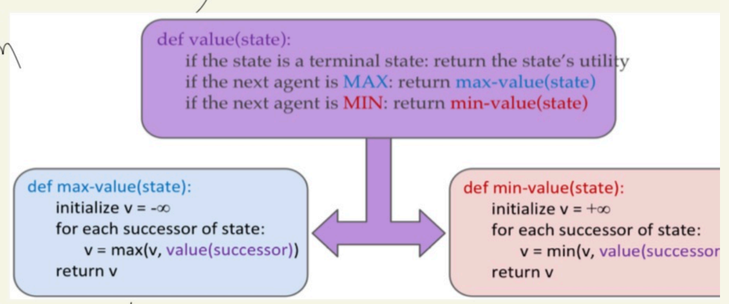
9. Local search
(1) no fringe, 是由完整的 option 去修改
(2) hill climbing ⇒ might stuck local optimum
(3) simulated annealing

```
function SIMULATED-ANNEALING(problem,schedule) returns a state
current ← problem.initial-state
for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.value – current.value
      if ΔE > 0 then current ← next
            else current ← next only with probability e^Δ
```

(4) local beam search
⇒ 同時跑k組 local search, 產生 successor, 找最好k個

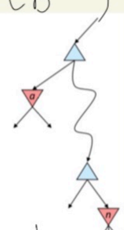
Chapter 4 adversarial search

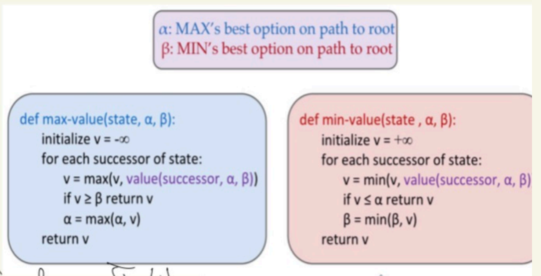
1. Formulation: states S (start s_0), players P_i, actions A, ...
2. 零和 game ⇒ maximize utility
3. Minimax computation

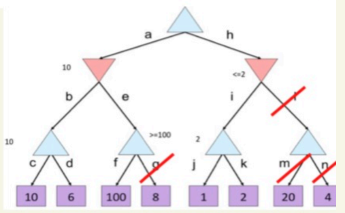


△ characteristics
⇒ 很像 DFS
⇒ time $O(b^m)$, space $O(bm)$
4. alpha beta pruning $O(b^{m/2})$

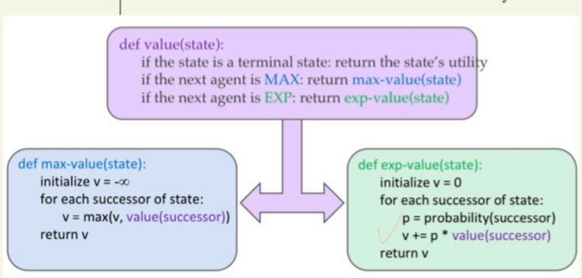
General configuration (MIN version)
- We're computing the MIN-VALUE at some node n
- We're looping over n's children
- n's estimate of the childrens' min is dropping
- Who cares about n's value? MAX
- Let a be the best value that MAX can get at any choice point along the current path from the root
- If n becomes worse than a, MAX will avoid it, so we can stop considering n's other children (it's already bad enough that it won't be played)







(1) if depth limited ⇒ 用 evaluation function 取代真實值.
(2) 有可能 starve ⇒ 如果 func. 不夠好 ⇒ 可能 迴圈
5. Expectimax: 對手 outcome by chance



(1) pruning? ⇒ 因為取 E[ ] ⇒ 無法 用一部分就捨棄

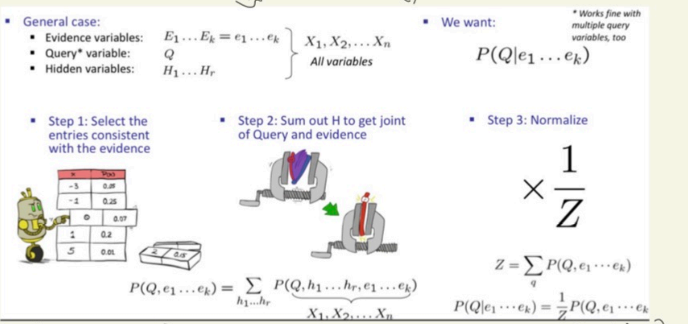
Chapter 5 Bayes

1. 機率 basics

(1) marginal distribution or conditional prob

$\Rightarrow P(x) = \sum_y P(x,y) \quad P(a|b) = \frac{P(a,b)}{P(b)}$

(2) Inference by enumeration



$O(d^n)$

(4) Product Rule

$P(X_1, X_2, X_3) = P(X_1) \times P(X_2|X_1) \times P(X_3|X_1, X_2)$

(5) Bayes Rule: $P(x|y) = \frac{P(y|x)}{P(y)} P(x)$

$(\because P(x,y) = P(x|y)P(y) = P(y|x)P(x))$

(5) Conditional Independence (for X, Y, Z) if $\forall x,y \; P(x|y,z) = P(x|z)$

X cond. indep. of Y given Z

2. Bayes Net (1) nodes: variables

arcs: direct influence from A → B

indicate cond. prob.

(2) semantics ① acyclic, directed ② $A_1 \cdots A_n \to X$ $P(X|A_1 \cdots A_n)$

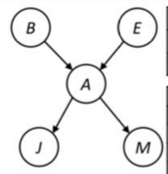
only depend on parent nodes

(3) $P(X_1, X_2, \dots X_n) = \prod_{i=1}^n P(X_i | \text{parents}(X_i))$

④ Note that Bayes Net 不代表因…

3. D-separation

e.g.



B	P(B)
+b	0.001
-b	0.999

E	P(E)
+e	0.002
-e	0.998

A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

$P(+b, -e, +a, -j, +m) =$
$P(+b)P(-e)P(+a|+b,-e)P(-j|+a)P(+m|+a) =$
$0.001 \times 0.998 \times 0.94 \times 0.1 \times 0.7$

(1) study independence on triples

① X → Y → Z X ind. of Z ✗

X ind Z given Y ✓

② Y; X ← Y → Z X ind Z ✗

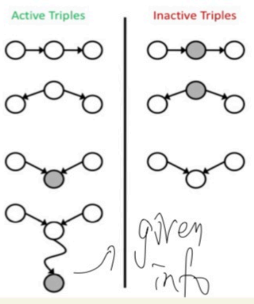
X ind Z given Y ✓

③ X → Z ← Y X ind Y ✓

X ind Y given Z ✗

(2) active / inactive path

- Question: Are X and Y conditionally independent given evidence variables {Z}?
  - Yes, if X and Y "d-separated" by Z
  - Consider all (undirected) paths from X to Y
  - No active paths = independence!
- A path is active if each triple is active:
  - Causal chain A -> B -> C where B is unobserved (either direction)
  - Common cause A <- B -> C where B is unobserved
  - Common effect (aka v-structure) A -> B <- C where B *or one of its descendants* is observed
- All it takes to block a path is a single inactive segment



- Query: $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$?
- Check all (undirected!) paths between X_i and X_j
  - If one or more active, then independence not guaranteed

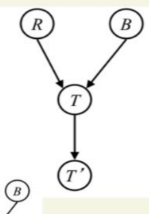
$X_i \not\perp\!\!\!\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$

  - Otherwise (i.e. if all paths are inactive), then independence is guaranteed

$X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$

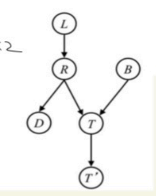
Ex1

$R \perp\!\!\!\perp B$ Yes
$R \perp\!\!\!\perp B | T$
$R \perp\!\!\!\perp B | T'$



Ex2

$L \perp\!\!\!\perp T' | T$ Yes
$L \perp\!\!\!\perp B$ Yes
$L \perp\!\!\!\perp B | T$
$L \perp\!\!\!\perp B | T'$
$L \perp\!\!\!\perp B | T, R$ Yes



4. Inference

(1) First basic operation: ***pointwise product*** of factors (similar to a ***database join***, ***not*** matrix multiply!)

- New factor has ***union*** of variables of the two original factors
- Each entry is the product of the corresponding entries from the original factors

Example: $P(J|A) \times P(A) = P(A,J)$

P(A)

true	0.1
false	0.9

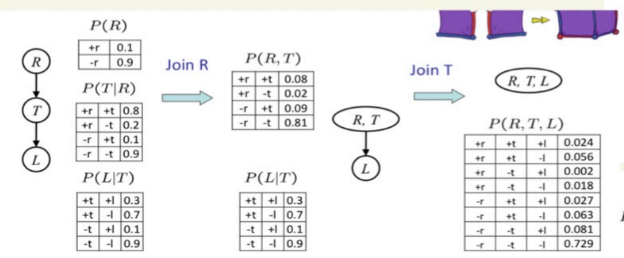
× P(J|A)

A \ J	true	false
true	0.9	0.1
false	0.05	0.95

= P(A,J)

A \ J	true	false
true	0.09	0.01
false	0.045	0.855

- Second basic operation: ***summing out*** (or eliminating) a variable from a factor
  - Shrinks a factor to a smaller one
- Example: $\sum_j P(A,J) = P(A,j) + P(A,\neg j) = P(A)$



P(R)

+r	0.1
-r	0.9

P(T|R)

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

P(L|T)

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

P(R,T)

+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

P(L|T)

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

P(R,T,L)

+r	+t	+l	0.024
+r	+t	-l	0.056
+r	-t	+l	0.002
+r	-t	-l	0.018
-r	+t	+l	0.027
-r	+t	-l	0.063
-r	-t	+l	0.081
-r	-t	-l	0.729

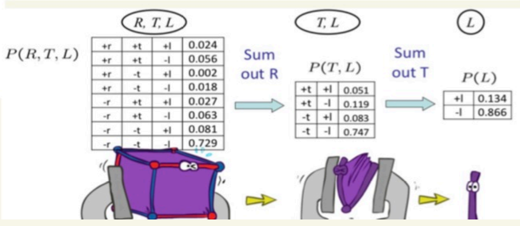
P(A,J)

A \ J	true	false
true	0.09	0.01
false	0.045	0.855

Sum out J

P(A)

true	0.1
false	0.9



P(R,T,L)

+r	+t	+l	0.024
+r	+t	-l	0.056
+r	-t	+l	0.002
+r	-t	-l	0.018
-r	+t	+l	0.027
-r	+t	-l	0.063
-r	-t	+l	0.081
-r	-t	-l	0.729

P(T,L)

+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

P(L)

+l	0.134
-l	0.866

Ch6 Sampling on Bayes Net

1. Prior Sampling

For i = 1, 2, …, n in topological order
Sample x_i from $P(X_i \mid \text{Parents}(X_i))$
Return $(x_1, x_2, \dots, x_n)$

2. Reject Sampling

⇒ prior sampling 但 reject 掉 不符 constraint 的

Input: evidence instantiation
For i = 1, 2, …, n in topological order
Sample x_i from $P(X_i \mid \text{Parents}(X_i))$
If x_i not consistent with evidence
Reject: return – no sample is generated in this cycle
Return $(x_1, x_2, \dots, x_n)$

3. Likelihood weighting: 想改進 sample 太多次

⇒ 直接 fix 過少的部分, 用 prob. 乘 weight

Input: evidence instantiation
w = 1.0
for i = 1, 2, …, n in topological order
if X_i is an evidence variable
X_i = observation x_i for X_i
Set w = w * $P(x_i \mid \text{Parents}(X_i))$
else
Sample x_i from $P(X_i \mid \text{Parents}(X_i))$
return $(x_1, x_2, \dots, x_n)$, w

Sampling distribution if z sampled and e fixed evidence

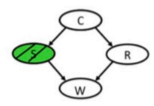
$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i|\text{Parents}(Z_i))$

Now, samples have weights

$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i|\text{Parents}(E_i))$

Together, weighted sampling distribution is consistent

$S_{WS}(z, e) \cdot w(z, e) = \prod_{i=1}^{l} P(z_i|\text{Parents}(z_i)) \prod_{i=1}^{m} P(e_i|\text{Parents}(e_i))$
$= P(\mathbf{z}, \mathbf{e})$



4. Gibbs sampling

⇒ 每次只選一個 free 其他 fix 做 sample (non-evidence)

5. Decision network: (choose action that max E[U] | evidence)

(1)

Umbrella = leave

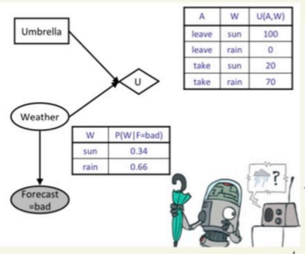
$EU(\text{leave}|\text{bad}) = \sum_w P(w|\text{bad})U(\text{leave}, w)$
$= 0.34 \cdot 100 + 0.66 \cdot 0 = 34$

Umbrella = take

$EU(\text{take}|\text{bad}) = \sum_w P(w|\text{bad})U(\text{take}, w)$
$= 0.34 \cdot 20 + 0.66 \cdot 70 = 53$

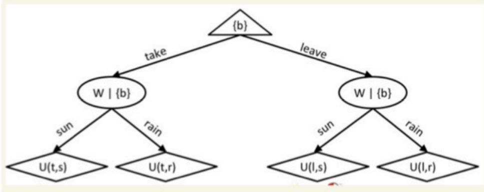
Optimal decision = take

$MEU(F = \text{bad}) = \max_a EU(a|\text{bad}) = 53$



A	W	U(A,W)
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

W	P(W|F=bad)
sun	0.34
rain	0.66



(2) Value of information: expected gain in MEU from new info

① $VPI(E'|e) = \left(\sum_{e'} P(e'|e) MEU(e,e')\right) - MEU(e)$

new info

嘗試 E' 的 $P(e'|e)$ ② a. $\forall E', e, \; VPI(E'|e) \geq 0$

b. non-additive; order indep.

6. Markov Models e.g. $X_1 \to X_2 \to \cdots X_n \dashrightarrow$ → $P(X_t)$ = ?

(1) Mini-Forward Alg. $P(X_1)$, $P(X_t|X_{t-1})$

$\Rightarrow P(x_t) = \sum_{x_{t-1}} P(x_{t-1}, x_t) = \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1})$

(2) ① 跟 $P(x_1)$ 長啥無關 ② $P_\infty(X) = P_{\infty+1}(X) = \cdots$

③ e.g.

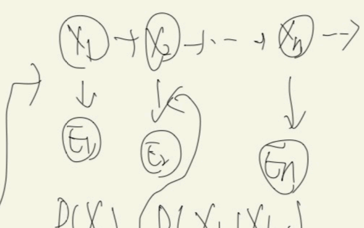
$P_\infty(sun) = P(sun|sun)P_\infty(sun) + P(sun|rain)P_\infty(rain)$
$P_\infty(rain) = P(rain|sun)P_\infty(sun) + P(rain|rain)P_\infty(rain)$

$P_\infty(sun) = 0.9P_\infty(sun) + 0.3P_\infty(rain)$
$P_\infty(rain) = 0.1P_\infty(sun) + 0.7P_\infty(rain)$

$P_\infty(sun) = 3P_\infty(rain)$
$P_\infty(rain) = 1/3P_\infty(sun)$

Also: $P_\infty(sun) + P_\infty(rain) = 1$ ⇒ $P_\infty(sun) = 3/4$, $P_\infty(rain) = 1/4$

7. Hidden Markov Model



$P(X_1)$, $P(X_t|X_{t-1})$, $P(E_t|X_t)$

(1) Find $P(X_t|e_{1:t})$ ($B(X_t)$)

$\Rightarrow P(X_{t+1}|e_{1:t}) = \sum_{x_t} P(X_{t+1}, x_t|e_{1:t}) = \sum P(X_{t+1}|x_t, e_{1:t}) P(x_t|e_{1:t})$

$B'(X_{t+1}) = \sum P(X_{t+1}|x_t) B(x_t) = \sum P(X_{t+1}|x_t) P(x_t|e_{1:t})$

$\Rightarrow P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, e_{t+1}|e_{1:t}) / P(e_{t+1}|e_{1:t})$

$= P(e_{t+1}|e_{1:t}, X_{t+1}) P(X_{t+1}|e_{1:t})$

$= P(e_{t+1}|X_{t+1}) B'(X_{t+1})$

We are given evidence at each time and want to know

$B_t(X) = P(X_t|e_{1:t})$

We can derive the following updates

$P(x_t|e_{1:t}) \propto_{X_t} P(x_t, e_{1:t})$
$= \sum_{x_{t-1}} P(x_{t-1}, x_t, e_{1:t})$
$= \sum_{x_{t-1}} P(x_{t-1}, e_{1:t-1}) P(x_t|x_{t-1}) P(e_t|x_t)$
$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})$

We can normalize as we go if we want to have P(x|e) at each time step, or just once at the end...