```
unify(T1,T2)
 S := {}
 stack := (T1 = T2)
 while stack is not empty
  pop (X = Y) from stack
  case
   [X is a variable, not occuring in Y]
   substitute Y for X in the stack and substitutions S
    S := S + {X/Y}
   [Y is a variable, not occuring in X]
   substitute X for Y in the stack and substitutions S
    S := S + {Y/X}
   [X and Y are identical constants or variables]
    nothing
   [X equals f(X1,X2,...Xn) and Y equals
f(Y1,Y2,...,Yn)]
    for i = 1..n
     push (Xi = Yi) onto stack
    rof
   [Else]
    return fail
  asac
 elihw
 return S
```

```
td _proof(Query)
 (T,S) := td_proof(Query,{})
return T
td_proof(Query,Subst)
 P := Query
 if (P is empty set)
  return true
 Q := atom element of P
 P := P - Q
 foreach rule R in KB
  (H :- B) := duplicate R with unique variables
  S := unify(Q,H)
  if (S is a substitution set) then
   Bs := B with substitution S applied.
   Ps := P with substitution S applied.
   SubstS := Subst with substitution S applied.
   Pb := Ps + Bs
   Ss := SubstS + S
   (T,S2) = td_proof(Pb,Ss)
   if (T is true) then
    return (true,S2)
   fi
  fi
 hcaerof
return (fail,{})
```

PHI: Constants -> D
PI: e.g. Pi(P)(b')=F
H. Universe: set of all ground constants
H. Base: set of all possible ground atoms for KB

Deterministic, fully observable: single-state
Single State:
  Initial state, goal test, successor fcn, path cost
  -Solution = sequence of actions: initial => goal
Non-observable: conformant / sensorless:
  -Devise a plan that works regardless of state (belief state)
Nondeterministic and/or partially ob.: contingency
Unknown state-space: exploration

Best-First search: Expand most desirable unexpanded node
Greedy Search: Not complete (can stuck in loops), $O(b^m)$ time and space, Not optimal.

A* Search: Complete (unless infinitely many nodes with $f \leq f(G)$, exponential time in relative error in h x length of solution, keep all nodes in memory.
Optimal. Why: $f(G_2)=g(G_2)>g(G_1)\geq f(n)$. Expand all nodes with $f(n)<C^*$ and some nodes with $f(n)=C^*$
f(n)=estimated total cost of path through n to goal
g(n)=cost so far to reach n
h(n)=estimated cost from n to goal

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes*	No	No	Yes

Min-Conflicts: Hill-climb with h(n)=total number of violated constraints.

CSP Heuristics:
1. Minimum Remaining Values (forward checking)
2. Most constraining variable
3. Least constraining value
4. Arc consistency (X->Y is consistent iff for every value x of X, there is some allowed y) $O(n^2d^2)$

Tree structured CSP: $O(d^c(n-c)d^2)$
Algorithm for tree structured CSPs:
1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
2. For j from n down to 2, apply RemoveInconsistent(Parent (Xj), Xj)
3. For j from 1 to n, assign Xj consistently with parent(Xj)

Minimax: Complete if finite, optimal against optimal opponent, $O(b^m)$ time, O(bm) depth-first exploration time. Improve with cutoff test and heuristic evaluation

function ALPHA-BETA-SEARCH(*state*, *game*) **returns** an action
  action, *state* ← the *a*, *s* **in** SUCCESSORS[*game*](*state*)
    such that MIN-VALUE($s, game, -\infty, +\infty$) is maximized
  return *action*

---

function MAX-VALUE($state, game, \alpha, \beta$) **returns** the minimax value of *state*
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow \max(\alpha, \text{MIN-VALUE}(s, game, \alpha, \beta))$
    if $\alpha \geq \beta$ **then return** β
  return α

---

function MIN-VALUE($state, game, \alpha, \beta$) **returns** the minimax value of *state*
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow \min(\beta, \text{MAX-VALUE}(s, game, \alpha, \beta))$
    if $\beta \leq \alpha$ **then return** α
  return β

Partial Order Planning:

function POP(*initial*, *goal*, *operators*) **returns** *plan*
  plan ← MAKE-MINIMAL-PLAN(*initial*, *goal*)
  loop do
    if SOLUTION?(*plan*) **then return** *plan*
    S_{need}, *c* ← SELECT-SUBGOAL(*plan*)
    CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, *c*)
    RESOLVE-THREATS(*plan*)
  end

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, *c*
  pick a plan step S_{need} from STEPS(*plan*)
    with a precondition *c* that has not been achieved
  return S_{need}, *c*

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, *c*)
  choose a step S_{add} from *operators* or STEPS(*plan*) that has *c* as an effect
  if there is no such step **then fail**
  add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
  add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
  if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

procedure RESOLVE-THREATS(*plan*)
  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
    choose either
      Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
      Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
    if not CONSISTENT(*plan*) **then fail**
  end

Exact Inferences
Time / Space cost = $O(d^k n)$ for polygraph; NP-hard otherwise

Performance element	Component	Representation	Feedback
Alpha–beta search	Eval. fn.	Weighted linear function	Win/loss
Logical agent	Transition model	Successor–state axioms	Outcome
Utility–based agent	Transition model	Dynamic Bayes net	Outcome
Simple reflex agent	Percept–action fn	Neural net	Correct action

Reference Sheet created by:
Vincent Chu
chuvincent (at) gmail (dot) com

Uncertainty
Pure logical approach risks falsehood, or lead to conclusions that have too many constraints. Probability summarizes laziness and ignorance, and can be generalized from previous experience
1) Boolean 2) Discrete random variables 3) Continuous

P(Y|E=e) = aP(Y, E=e) = $a\sum_h$(Y, E=e, H=h)

$$\begin{aligned}&\mathbf{P}(Cavity|toothache) = \alpha\,\mathbf{P}(Cavity, toothache)\\&= \alpha\,[\mathbf{P}(Cavity, toothache, catch) + \mathbf{P}(Cavity, toothache, \neg catch)]\\&= \alpha\,[\langle 0.108, 0.016\rangle + \langle 0.012, 0.064\rangle]\\&= \alpha\,\langle 0.12, 0.08\rangle = \langle 0.6, 0.4\rangle\end{aligned}$$

$O(d^n)$ worst time, and space

Bayes Rule:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

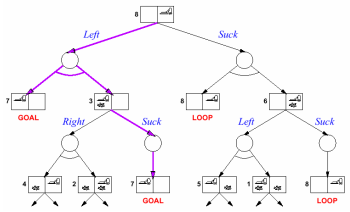
Planning:
Divide and conquer by sub-goaling

States	Logical Sentence
Action	Preconditions/outcomes
Goal	Logical Sentence (Conjuction)
Plan	Constraint on action

STRIPS operator: PRECONDITION, ACTION, EFFECT

Problem with partial order planning:
=Incomplete Information:
  -Unknown preconditions
  -Multiple effects
=Incorrect information (Missing / incorrect postconditions in operators)
=Qualification problem

Other types of planning:
=Conditional Planning:
 -Plan to obtain info (splits up belief state) + subplan for each contingency (if then else)



=Monitoring / Replanning:
-Keep checking all preconditions of remaining steps, all causal links crossing current time point

Conjunctive Query:

$$\mathbf{P}(X_i, X_j|\mathbf{E}=\mathbf{e}) = \mathbf{P}(X_i|\mathbf{E}=\mathbf{e})\mathbf{P}(X_j|X_i, \mathbf{E}=\mathbf{e})$$

Continuous Variable Formulae:

$$\begin{aligned}&P(Cost=c|Harvest=h, Subsidy?=true)\\&= N(a_t h + b_t, \sigma_t)(c)\\&= \frac{1}{\sigma_t\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{c-(a_t h+b_t)}{\sigma_t}\right)^2\right)\end{aligned}$$

$$P(Buys?=true \mid Cost=c) = \frac{1}{1+exp(-2\frac{-c+\mu}{\sigma})}$$

Conditional Independence
A is conditonally independent of B given C
P(A|B,C)=P(A|C) <=>
P(B|A,C)=P(B|C) <=>
P(A,B|C)=P(A|C)P(B|C)

Conditional Indepence reduces the size of joint distribution from exponential in n to linear in n.

Conditional Independence in Baye's Net
-Each node is conditionally independent of its nondescendants given its parents
-Each node is cond. independent of all others given its parents + children + children's parents
-P(X|E)… node Y is irrelevant unless
Y∈ Ancestors({X} ∪ E)

Moral Graph – Mary all parents & drop arrows
=> Y is irrelevant if it's m-separated from X by E

Constraints:

Orderability
$(A \succ B) \vee (B \succ A) \vee (A \sim B)$

Transitivity
$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

Continuity
$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1-p, C] \sim B$

Substitutability
$A \sim B \Rightarrow [p, A;\ 1-p, C] \sim [p, B; 1-p, C]$

Monotonicity
$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succsim [q, A;\ 1-q, B])$

Value of information

$$VPI_E(E_j) = \left(\Sigma_k\ P(E_j = e_{jk}|E) EU(\alpha_{e_{jk}}|E, E_j = e_{jk})\right) - EU(\alpha|E)$$
$$EU(\alpha_{e_{jk}}|E, E_j = e_{jk}) = \max_a \Sigma_i\ U(S_i)\ P(S_i|E, a, E_j = e_{jk})$$

VPI is nonnegative, nonadditive and order-indepedent

Learning – System construction & designer lacks omniscience
Depends on type of performance element used, functional component learnt and its representation, and what feedback.
-Adjust hypothesis h to agree with f
(h is consistent when it agrees with f on all samples)
-Ockham's razor: maximize consistency and simplicity

Decision Tree Learning

function DTL(*examples, attributes, default*) **returns** a decision tree

if *examples* is empty **then return** *default*
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** MODE(*examples*)
else
  best ← CHOOSE-ATTRIBUTE(*attributes, examples*)
  tree ← a new decision tree with root test *best*
  for each value v_i of *best* **do**
    $examples_i$ ← {elements of *examples* with *best* = v_i}
    subtree ← DTL($examples_i$, *attributes* − *best*, MODE(*examples*))
    add a branch to *tree* with label v_i and subtree *subtree*
  return *tree*

Choose Attribute:

$$\Sigma_i\ \frac{p_i + n_i}{p + n} H(\langle p_i/(p_i + n_i), n_i/(p_i + n_i)\rangle)$$
$$H(\langle P_1, \ldots, P_n\rangle) = \Sigma_{i=1}^{n} - P_i \log_2 P_i$$

Neural Network (Perceptron) $g=(1+e^{-x})^{-1}$

Perceptron

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\Sigma_{j=0}^{n} W_j x_j)\right)$$
$$= -Err \times g'(in) \times x_j$$

```
W = random initial values
for iter = 1 to T
    for i = 1 to N (all examples)
        x = input for example i
        y = output for example i
        W_old = W
        Err = y − g(W_old · x)
        for j = 1 to M (all weights)
            W_j = W_j + α · Err · g'(W_old · x) · x_j
```

Temporal Probability Network

-Markov assumption: x_i depends on bounded subset of $x_{0:t-1}$
-Sensor Markov assumption: $P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t)$
-Stationary Process: $P(X_t|X_{t-1})$ and $P(E_t|X_t)$ are the same for all t

Filtering $P(X_t \mid e_{1:t})$

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$$
$$= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \Sigma_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t|\mathbf{e}_{1:t}) \text{ Condition on } X_t$$
$$= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \Sigma_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) P(\mathbf{x}_t|\mathbf{e}_{1:t}) \text{ Markov assumption}$$

$$\mathbf{f}_{1:t+1} = \text{FORWARD}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1}) \text{ where } \mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$$
Time and space constant (independent of t)

Smoothing $P(X_k \mid e_{1:t})$ $0 \leq k \leq t$

$$\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) = \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t}$$
$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \Sigma_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \text{ Condition on } X_{k+1}$$
$$= \Sigma_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \text{ Cond. ind.}$$
$$= \Sigma_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \text{ Cond. ind.}$$

Most likely explanation $\mathbf{argmax}_{x1:t} P(x_{1:t} \mid e_{1:t})$

$$\max_{\mathbf{x}_1 \ldots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{X}_{t+1}|\mathbf{e}_{1:t+1})$$
$$= \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left(\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t|\mathbf{e}_{1:t})\right)$$

Stochastic
Rejection Sampling – Expensive if P(e) is small. (Drops off exponentially with number of evidence variables)
Likelihood weighting – Fix evidence variables, sample only non-evidence variables and weight each sample by the likelihood

Neural Network (Back Propagation)

```
for iter = 1 to T
  for e = 1 to N (all examples)
    x = input for example e
    y = output for example e
    run x forward through network, computing all {a_i}, {in_i}
    for all weights (j,i) (in reverse order)
      compute Δ_i = { (y_i − a_i) × g'(in_i)   if i is output node
                    { g'(in_i) Σ_k W_{i,k} Δ_k        o.w.
      W_{j,i} = W_{j,i} + α × a_j × Δ_i
```

Vision – Inverse Graphics, P(W|S)=P(S|W)P(W) (Graphics x Prior Knowledge)
Edges provide 1) depth 2) surface orientation 3)reflectance(surface marks) 4)illumination (shadows)
Cues: 1) motion 2) sterero 3) texture 4) shading 5)contour
Shape Context matching: bullseye + histogram => dist. b/w shapes = sum of dist. b/w corresponding pts.
Robotics – 6 degrees of freedom. |DOF| != |Controls|
Configuration Space Planning: 1)Recursive Cell decomposition
2)Skeletonization: Locus of points equidistance from obstacles (Voronoi Diagram)

```
function MAX-VALUE(state,alpha,beta) returns a utility value
 inputs: state, current state in game
       alpha, the value of the best alternative for MAX along the path to state
       beta, the value of the best alternative for MIN along the path to state

 if TERMINAL-TEST(state) then
  return UTILITY(state)
 else
  v <- h_min
  for a,s in SUCCESSORS(state) do
   v <- MAX(v,CHANCE-MAX(s,alpha,beta)
   if v >= beta then return v
   alpha <- MAX(alpha,v)
  end
  return v
 end

function CHANCE-MAX(state,alpha,beta) returns a utility value
 inputs: state, current state in game
       alpha, the value of the best alternative for MAX along the path to state
       beta, the value of the best alternative for MIN along the path to state
 if TERMINAL-TEST(state) then
  return UTILITY(state)
 else
  v <- array of zeros
  u_bound = UPPER_BOUND(v,0)
  l_bound = LOWER_BOUND(v,0)

  for i = 1 to length(rolls) do
   s <- state with dice roll set to be rolls(i)
   alpha_prime = COMPUTE_AP(alpha,u_bound,i)
   beta_prime = COMPUTE_BP(beta,l_bound,i)
   v(i) = MIN-VALUE(s,alpha_prime,beta_prime)
   u_bound = UPPER_BOUND(v,i)
   l_bound = LOWER_BOUND(v,i)
   if u_bound <= alpha then return u_bound
  end
  return UPPER_BOUND(v,length(rolls))  // This isn't a bound, but an exact value.
 end

function UPPER_BOUND(v,i) returns an upper bound
 inputs: v, array of evaluated node values
       i, index of last evaluated node

 u_bound = 0;
 for j = 1 to i do
  u_bound = u_bound + v(j)*roll_probs(j)
 end
 u_bound = u_bound + (v(i+1)+v(i+2)+...+v(length(roll_probs)))*h_max
 return u_bound
```