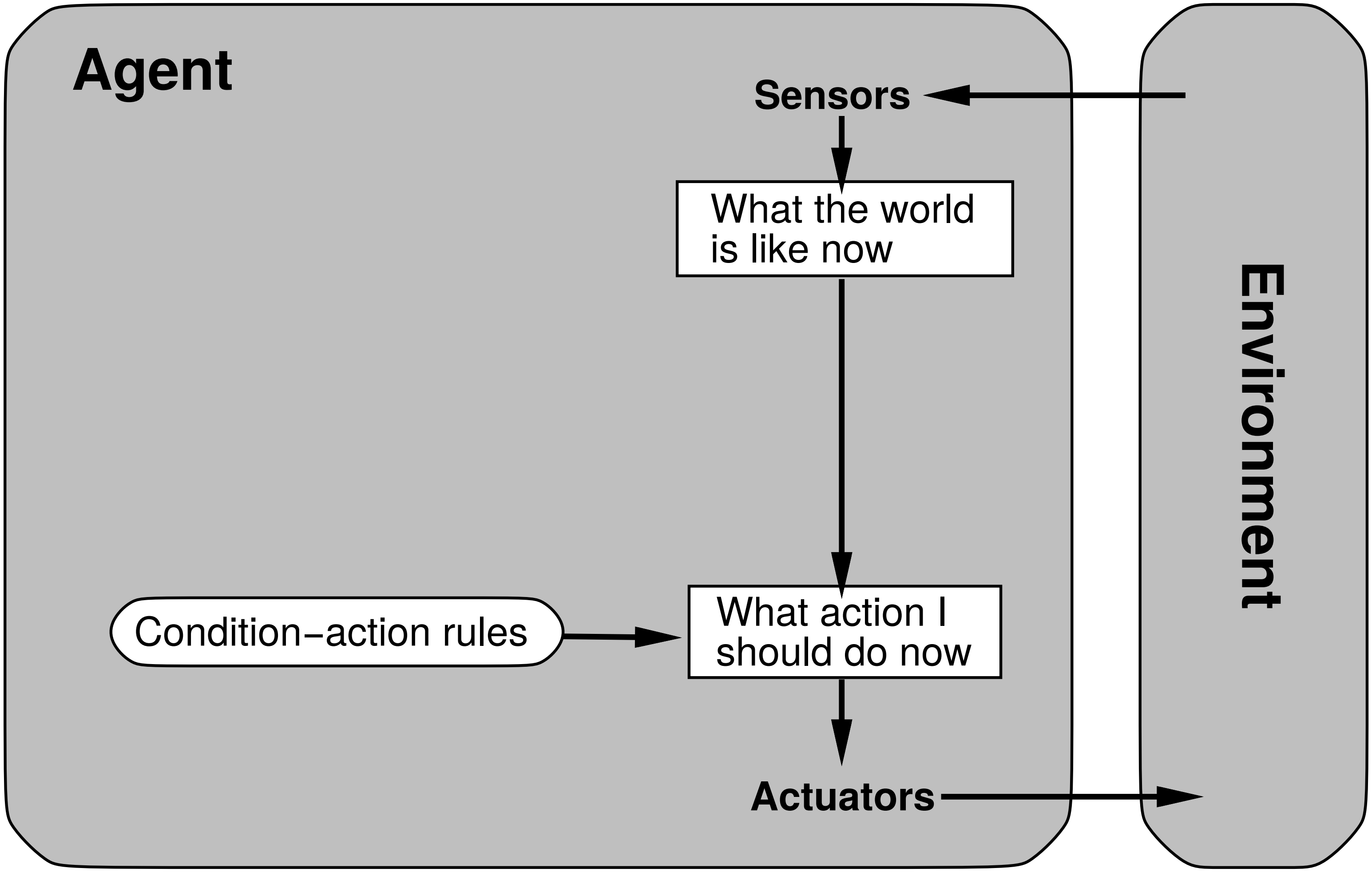
Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No
Single-agent??	Yes	No	Yes (except auctions)	No

The environment type largely determines the agent design

The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Simple reflex agents

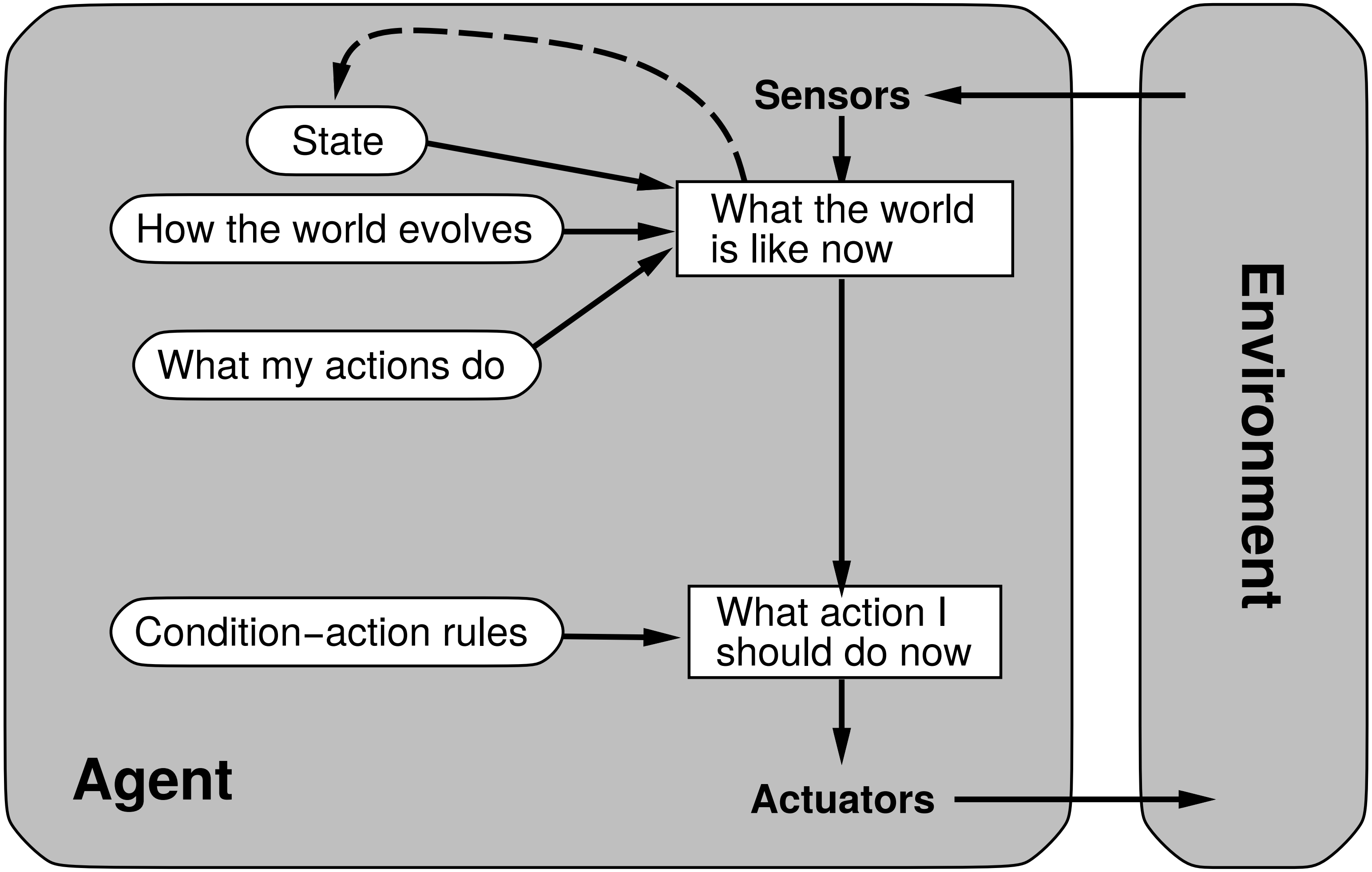


Example

```
function REFLEX-VACUUM-AGENT( [$location$,$status$]) returns an action
    if $status = Dirty$ then return $Suck$
    else if $location = A$ then return $Right$
    else if $location = B$ then return $Left$
```

```
(setq joe (make-agent :name 'joe :body (make-agent-body)
                      :program (make-reflex-vacuum-agent-program))

(defun make-reflex-vacuum-agent-program ()
  #'(lambda (percept)
      (let ((location (first percept)) (status (second percept)))
        (cond ((eq status 'dirty) 'Suck)
              ((eq location 'A) 'Right)
              ((eq location 'B) 'Left)))))
```

Reflex agents with state

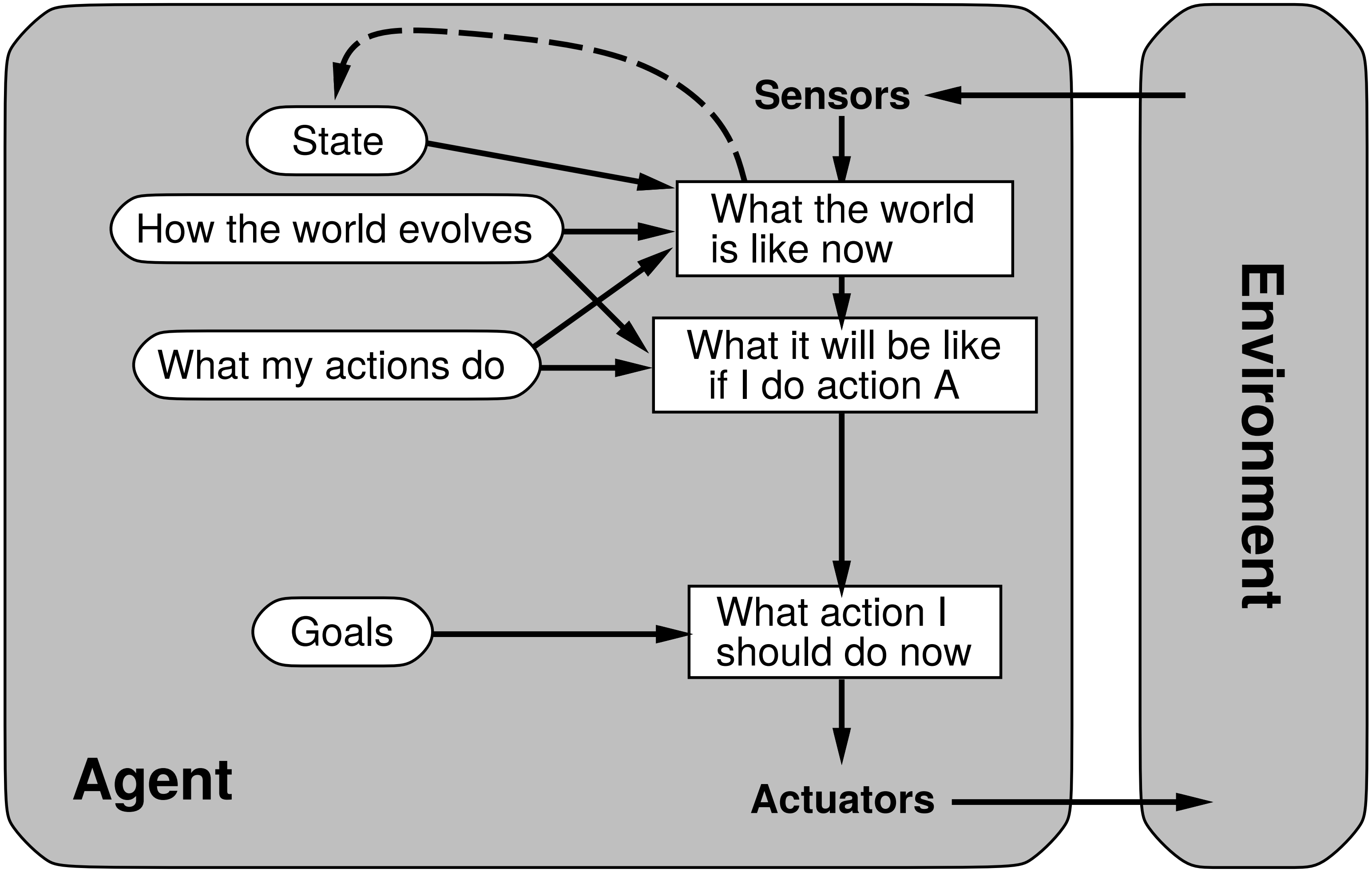


Example

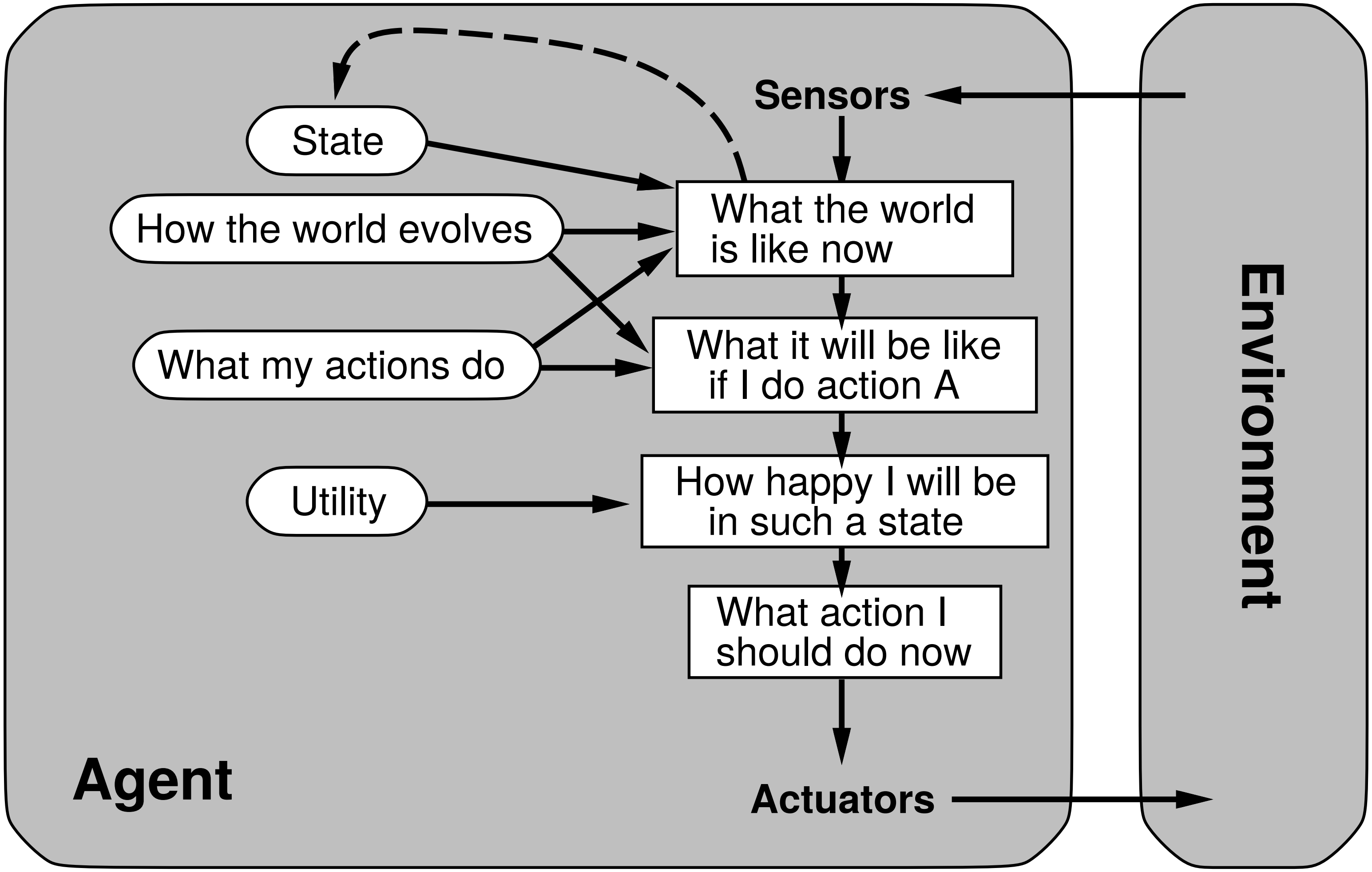
```
function REFLEX-VACUUM-AGENT( [$location$,$status$]) returns an action
static: $last\_A$, $last\_B$, numbers, initially $\infty$

    if $status = Dirty$ then . . .
```

```
(defun make-reflex-vacuum-agent-with-state-program ()
  (let ((last-A infinity) (last-B infinity))
  #'(lambda (percept)
      (let ((location (first percept)) (status (second percept)))
        (incf last-A) (incf last-B)
        (cond
         ((eq status 'dirty)
          (if (eq location 'A) (setq last-A 0) (setq last-B 0))
          'Suck)
         ((eq location 'A) (if (> last-B 3) 'Right 'NoOp))
         ((eq location 'B) (if (> last-A 3) 'Left 'NoOp)))))))
```

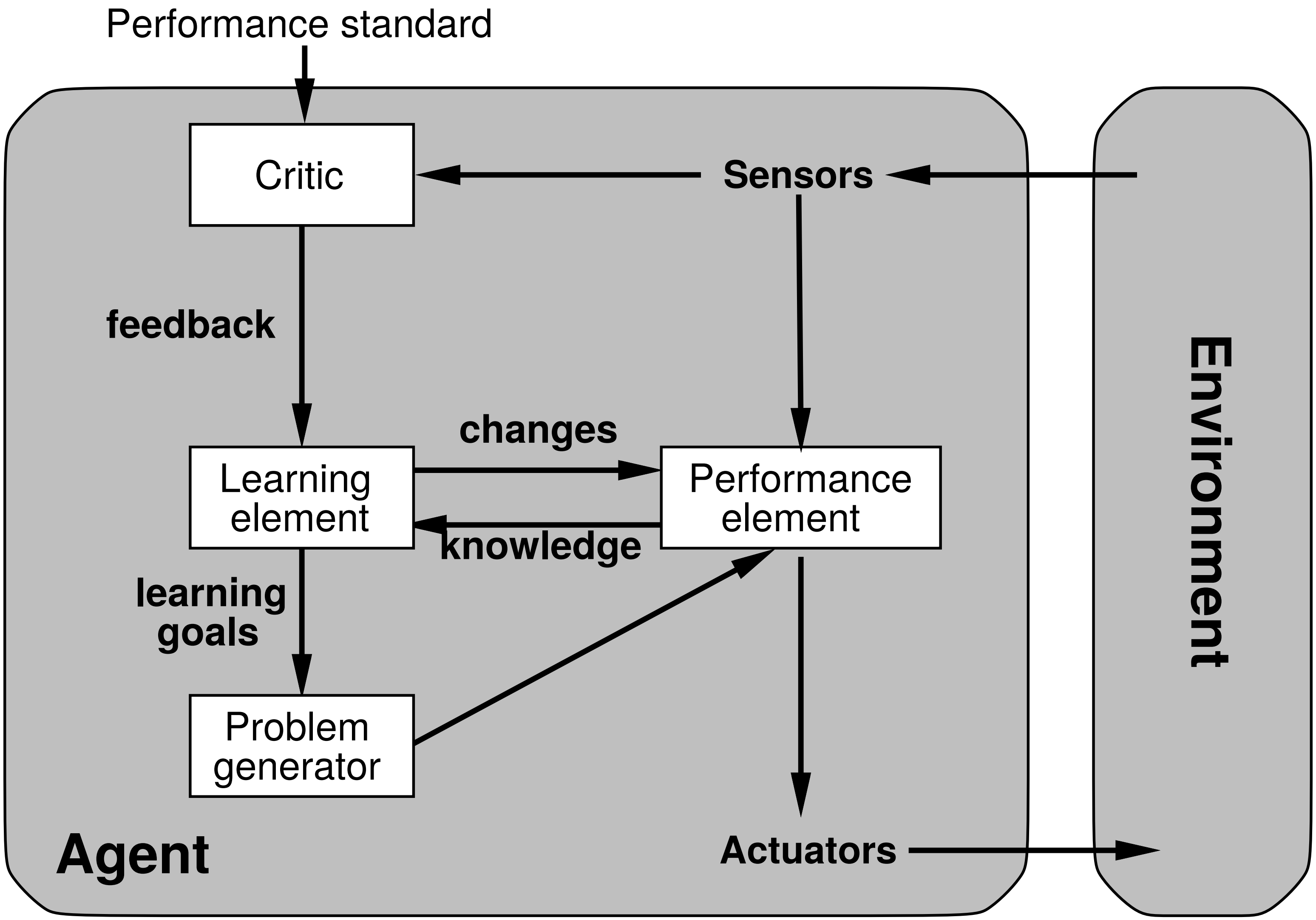
Goal-based agents


Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Utility-based agents


State
How the world evolves
What my actions do
Utility
Sensors
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

Learning agents



Summary

Agents interact with environments through actuators and sensors

The agent function describes what the agent does in all circumstances

The performance measure evaluates the environment sequence

A perfectly rational agent maximizes expected performance

Agent programs implement (some) agent functions

PEAS descriptions define task environments

Environments are categorized along several dimensions:
observable? deterministic? episodic? static? discrete? single-agent?

Several basic agent architectures exist:
reflex, reflex with state, goal-based, utility-based

Tree search algorithms

Basic idea:

offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. expanding states)

```
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
```

Implementation: general tree search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    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
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Search strategies

A strategy is defined by picking the **order of node expansion**

Strategies are evaluated along the following dimensions:
- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory
- optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of
- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? Yes (if cost $= 1$ per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs = 8640GB.

Uniform-cost search

Expand least-cost unexpanded node

Implementation:

$fringe =$ queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal?? No

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

```
function Depth-Limited-Search( problem, limit) returns soln/fail/cutoff
    Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit)

function Recursive-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? $\leftarrow$ false
    if Goal-Test(problem, State[node]) then return node
    else if Depth[node] $=$ limit then return cutoff
    else for each successor in Expand(node, problem) do
        result $\leftarrow$ Recursive-DLS(successor, problem, limit)
        if result $=$ cutoff then cutoff-occurred? $\leftarrow$ true
        else if result $\neq$ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

```
function Iterative-Deepening-Search( problem) returns a solution
    inputs: problem, a problem

    for depth $\leftarrow$ 0 to $\infty$ do
        result $\leftarrow$ Depth-Limited-Search( problem, depth)
        if result $\neq$ cutoff then return result
    end
```

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost = 1
Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$\begin{aligned} N(\mathsf{IDS}) &= 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450 \\ N(\mathsf{BFS}) &= 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100 \end{aligned}$$

IDS does better because other nodes at depth d are not expanded

BFS can be modified to apply goal test when a node is **generated**

Summary of algorithms

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*

Graph search

```
function Graph-Search( problem, fringe) returns 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
            fringe ← InsertAll(Expand(node, problem), fringe)
    end
```

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h × length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

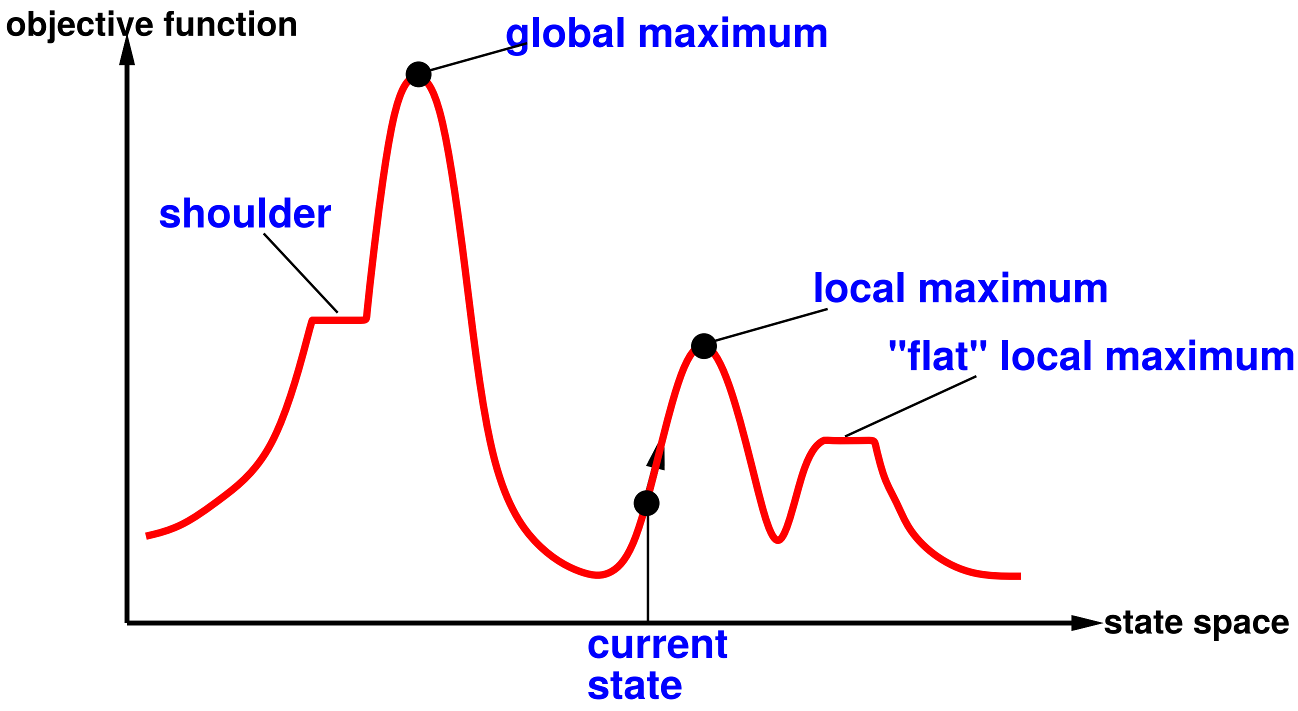
Hill-climbing (or gradient ascent/descent)

“Like climbing Everest in thick fog with amnesia”

```
function Hill-Climbing( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← Make-Node(Initial-State[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if Value[neighbor] $\leq$ Value[current] then return State[current]
        current ← neighbor
    end
```

Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😊 escape from shoulders ☹ loop on flat maxima

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     $T$, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for $t \leftarrow 1$ to $\infty$ do
        $T \leftarrow$ schedule[$t$]
        if $T = 0$ then return current
        next ← a randomly selected successor of current
        $\Delta E \leftarrow$ VALUE[next] – VALUE[current]
        if $\Delta E > 0$ then current ← next
        else current ← next only with probability $e^{\Delta E/T}$
```

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^*
because $e^{\frac{E(x^*)}{kT}}/e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Local beam search

Idea: keep k states instead of 1; choose top k of all their successors

Not the same as k searches run in parallel!
Searches that find good states recruit other searches to join them

Problem: quite often, all k states end up on same local hill

Idea: choose k successors randomly, biased towards good ones

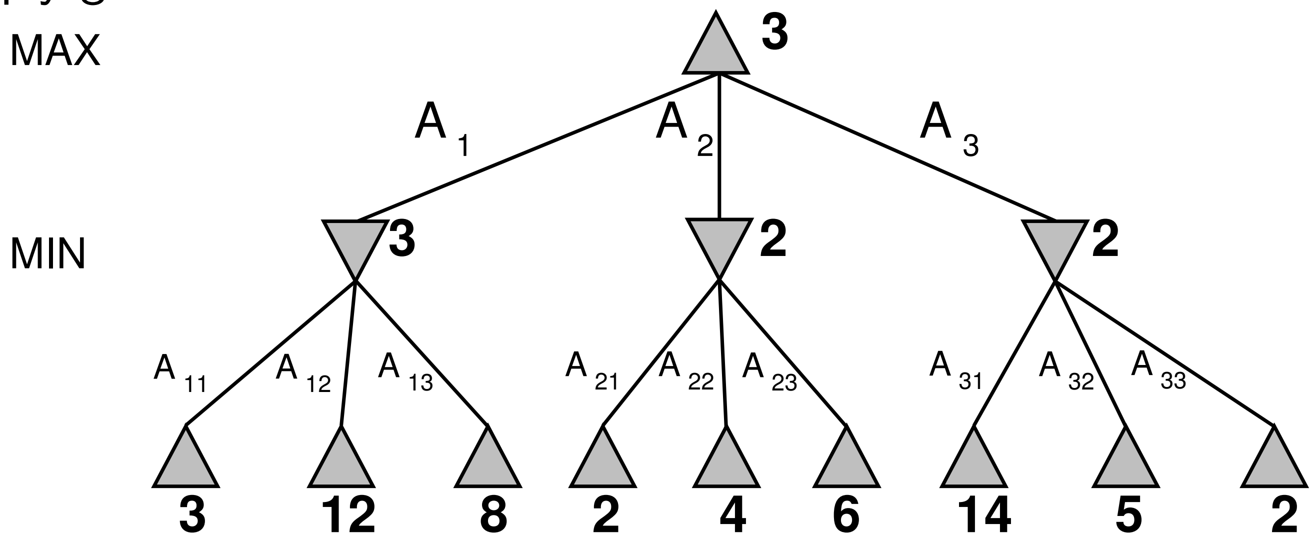
Observe the close analogy to natural selection!

Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

E.g., 2-ply game:



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for a, s in SUCCESSORS(state) do $v \leftarrow$ MAX($v$, MIN-VALUE(s))
    return $v$

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow \infty$
    for a, s in SUCCESSORS(state) do $v \leftarrow$ MIN($v$, MAX-VALUE(s))
    return $v$
```

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$\Rightarrow$ exact solution completely infeasible

But do we need to explore every path?

The α–β algorithm

```
function Alpha-Beta-Decision(state) returns an action
    return the a in Actions(state) maximizing Min-Value(Result(a, state))
```

```
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)
    $v \leftarrow -\infty$
    for a, s in Successors(state) do
        $v \leftarrow$ Max($v$, Min-Value($s, \alpha, \beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ Max($\alpha$, $v$)
    return $v$
```

```
function Min-Value(state, $\alpha$, $\beta$) returns a utility value
    same as Max-Value but with roles of $\alpha, \beta$ reversed
```

Properties of α–β

Pruning **does not** affect final result

Good move ordering improves effectiveness of pruning

With “perfect ordering,” time complexity $= O(b^{m/2})$
$\Rightarrow$ **doubles** solvable depth

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Unfortunately, 35^{50} is still impossible!

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game*

Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

Special case: if an action is optimal for all deals, it's optimal.*

GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

Entailment

Entailment means that one thing **follows from** another:

$$KB \models \alpha$$

Knowledge base KB entails sentence α
if and only if
α is true in all worlds where KB is true

E.g., the KB containing “the Giants won” and “the Reds won” entails “Either the Giants won or the Reds won”

E.g., $x + y = 4$ entails $4 = x + y$

Entailment is a relationship between sentences (i.e., **syntax**) that is based on **semantics**

Note: brains process **syntax** (of some sort)

Models

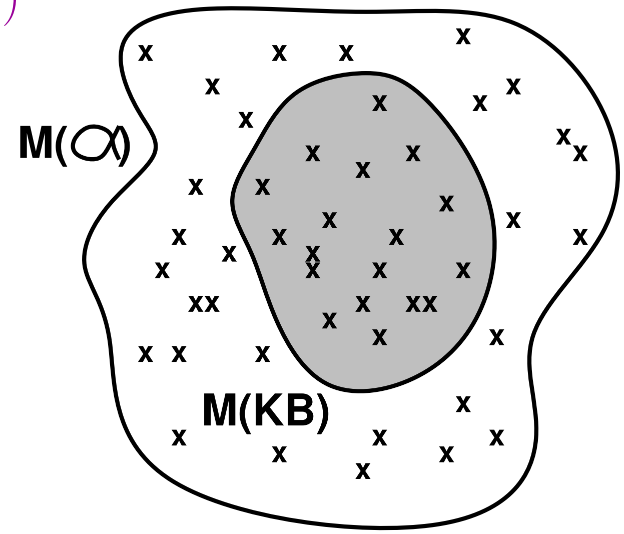
Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

We say m is a model of a sentence α if α is true in m

$M(\alpha)$ is the set of all models of α

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

E.g. $KB =$ Giants won and Reds won
$\alpha =$ Giants won



Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment = needle in haystack; inference = finding it

Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Logical equivalence

Two sentences are logically equivalent iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$\begin{aligned}
(\alpha \land \beta) &\equiv (\beta \land \alpha) && \text{commutativity of } \land \\
(\alpha \lor \beta) &\equiv (\beta \lor \alpha) && \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) &\equiv (\alpha \land (\beta \land \gamma)) && \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) &\equiv (\alpha \lor (\beta \lor \gamma)) && \text{associativity of } \lor \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \;\Rightarrow\; \beta) &\equiv (\neg\beta \;\Rightarrow\; \neg\alpha) && \text{contraposition} \\
(\alpha \;\Rightarrow\; \beta) &\equiv (\neg\alpha \lor \beta) && \text{implication elimination} \\
(\alpha \;\Leftrightarrow\; \beta) &\equiv ((\alpha \;\Rightarrow\; \beta) \land (\beta \;\Rightarrow\; \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \land \beta) &\equiv (\neg\alpha \lor \neg\beta) && \text{De Morgan} \\
\neg(\alpha \lor \beta) &\equiv (\neg\alpha \land \neg\beta) && \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) &\equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) && \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) &\equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) && \text{distributivity of } \lor \text{ over } \land
\end{aligned}$$

Pros and cons of propositional logic

😊 Propositional logic is **declarative**: pieces of syntax correspond to facts

😊 Propositional logic allows partial/disjunctive/negated information
(unlike most data structures and databases)

😊 Propositional logic is **compositional**:
meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

😊 Meaning in propositional logic is **context-independent**
(unlike natural language, where meaning depends on context)

☹ Propositional logic has very limited expressive power
(unlike natural language)
E.g., cannot say "pits cause breezes in adjacent squares"
except by writing one sentence for each square

First-order logic

Whereas propositional logic assumes world contains **facts**,
first-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
- Relations: red, round, bogus, prime, multistoried . . .,
  brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, . . .
- Functions: father of, best friend, third inning of, one more than, end of . . .

Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB
and perceives a smell and a breeze (but no glitter) at $t = 5$:

$Tell(KB, Percept([Smell, Breeze, None], 5))$
$Ask(KB, \exists\, a \;\; Action(a, 5))$

I.e., does KB entail any particular actions at $t = 5$?

Answer: $Yes,\ \{a/Shoot\}$ ← substitution (binding list)

Given a sentence S and a substitution σ,
$S\sigma$ denotes the result of plugging σ into S; e.g.,
$S = Smarter(x, y)$
$\sigma = \{x/Hillary, y/Bill\}$
$S\sigma = Smarter(Hillary, Bill)$

$Ask(KB, S)$ returns some/all σ such that $KB \models S\sigma$

Knowledge base for the wumpus world

"Perception"
$\forall\, b, g, t \;\; Percept([Smell, b, g], t) \;\Rightarrow\; Smelt(t)$
$\forall\, s, b, t \;\; Percept([s, b, Glitter], t) \;\Rightarrow\; AtGold(t)$

Reflex: $\forall\, t \;\; AtGold(t) \;\Rightarrow\; Action(Grab, t)$

Reflex with internal state: do we have the gold already?
$\forall\, t \;\; AtGold(t) \wedge \neg Holding(Gold, t) \;\Rightarrow\; Action(Grab, t)$

$Holding(Gold, t)$ cannot be observed
⇒ keeping track of change is essential

Deducing hidden properties

Properties of locations:

$\forall\, x, t\;\; At(Agent, x, t) \wedge Smelt(t) \;\Rightarrow\; Smelly(x)$

$\forall\, x, t\;\; At(Agent, x, t) \wedge Breeze(t) \;\Rightarrow\; Breezy(x)$

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect

$$\forall\, y\;\; Breezy(y) \;\Rightarrow\; \exists\, x\;\; Pit(x) \wedge Adjacent(x, y)$$

Causal rule—infer effect from cause

$$\forall\, x, y\;\; Pit(x) \wedge Adjacent(x, y) \;\Rightarrow\; Breezy(y)$$

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the $Breezy$ predicate:

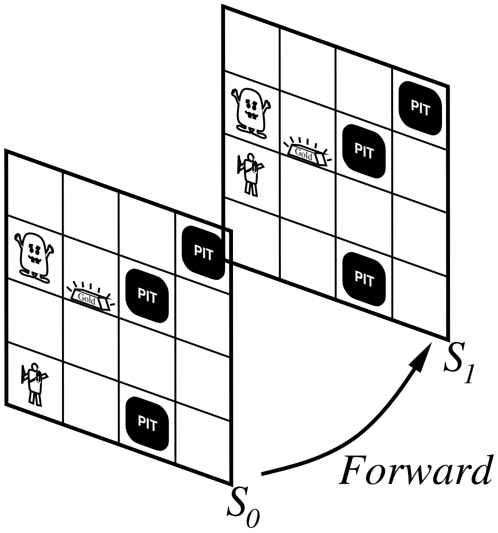
$$\forall\, y\;\; Breezy(y) \;\Leftrightarrow\; [\exists\, x\;\; Pit(x) \wedge Adjacent(x, y)]$$

Keeping track of change

Facts hold in situations, rather than eternally
E.g., $Holding(Gold, Now)$ rather than just $Holding(Gold)$

Situation calculus is one way to represent change in FOL:
Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a in s



Describing actions I

“Effect” axiom—describe changes due to action
$\forall\, s \;\; AtGold(s) \;\Rightarrow\; Holding(Gold, Result(Grab, s))$

“Frame” axiom—describe **non-changes** due to action
$\forall\, s \;\; HaveArrow(s) \;\Rightarrow\; HaveArrow(Result(Grab, s))$

Frame problem: find an elegant way to handle non-change
(a) representation—avoid frame axioms
(b) inference—avoid repeated “copy-overs” to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, . . .

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is "about" a **predicate** (not an action per se):

$$\begin{array}{lcl} \text{P true afterwards} & \Leftrightarrow & [\text{an action made P true} \\ & \lor & \text{P true already and no action made P false}] \end{array}$$

For holding the gold:

$$\forall\, a, s \;\; Holding(Gold, Result(a, s)) \;\Leftrightarrow$$
$$[(a = Grab \land AtGold(s))$$
$$\lor\, (Holding(Gold, s) \land a \neq Release)]$$

Making plans

Initial condition in KB:

$At(Agent, [1,1], S_0)$
$At(Gold, [1,2], S_0)$

Query: $Ask(KB, \exists\, s\;\; Holding(Gold, s))$
i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$
i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $Ask(KB, \exists\, p\ \ Holding(Gold, PlanResult(p, S_0)))$
has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$$\forall\, s\ \ PlanResult([\,], s) = s$$
$$\forall\, a, p, s\ \ PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Introduction to Artificial Inteligence

Example Quiz 1 - answers (currently being edited!)

Justify all answers CONCISELY!

1) a) Number of people on the wrong shore is an admissible heuristic
is correct - value is 0 for goal, and we always need to move at least
as many people as there are on the wrong shore (note the PATH COST is
NUMBER OF PEOPLE moved, not number of trips!)

2) Construct a taxonomic hierarchy for animals, which contains mammals, which
   contain primates, which in turn contains apes and humans (which are
   disjoint). Whales are
   also mammals, which are not primates, and no primate is a whale. All whales
   can swim. Jesus is a human who can walk on water, but cannot swim.
   Anything that can swim has a fin. Juju is a whale.

a)

```
animals - mammals - primates - humans (-) Jesus
                  \          - apes
                    whales            (-) Juju
```

axioms:

```
(forall x) x in mammals => x in animals  ; etc.
disjoint({whales, primates}, mammals)
(forall x) x in whales => swims(x)
(forall x) swims(x) => (exists y) fin(y) and part_of(y, x)

        fin(x) - true if object is a fin
        part_of(x, y) - true if object x is a part of object y
```

facts:

```
Jesus in humans.
can_walk_on_water(Jesus)  ; predicate signifying that object can walk on water
~swims(Jesus)             ; swims - predicate signifying object can swim
Juju in whales
```

b) (outlines only - full proof as in answer to exercises)

```
      1) Jesus has no fin           cannot be proved or disproved
      2) Jesus is a whale           disprove via inheritence and disjointness
                                    of whales and primates
      3) Some animals can swim      proved via example of Juju the whale,,
                                    thus a mammal, and thus an animal
      4) Some primates cannot swim  proved via example of Jesus who's a human,
                                    and thus a primate
      5) If some primate can walk on water, then there are no primates
                                    false - proved by counterexample (Jesus)
```

3) a) Idea is similar to MINIMAX. In general, for an n-person game
(value, best_value are lists of n elements):

```
best(position, #_of_players, player, depth) {
   if(terminal(position))
     return(value(position));
   best_value[player] = -infinity; best_move = error;
   foreach(move in legal_moves(position, player)) {
       new_position = apply_move(position, move);
       new_value = best(new_position, #_of_players,
                        (player+1) mod #_of_players, depth + 1);
       if(new_value[player] >= best_value[player]) {
          best_value = new_value;
          best_move = move;
       }
     }
     if(depth == 0)
        do_move(best_move);
     else
        return(best_value);
}
```

For b and c, let us name the moves: A's moves are a1 or a2, B's are b1 or b2, and C's c1 or c2.

b) If two players, say B and C, could make deals, this could lead to a a different choice of moves and possibly a worse score for A. In addition, A's algorithm would depend on HOW B and C make their deals. For example, in the following, B can agree with C NOT to make move b1 if C agrees NOT to make move c1. In this case BOTH B AND C benefit. This will NOT work without a prior agreement - if each tries direct maximization of individual score, the best they can hope for is 2 each.

```
0 - a1 - b1 - c1     (5 2 7)
 \     \    - c2     (5 7 2)
  \      b2 - c1     (5 0 7)
   \        - c2     (0 5 5)
    a2 - b1 - c1     (2 3 3)
       \    - c2     (2 3 4)
         b2 - c1     (2 0 0)
            - c2     (2 0 0)
```

As a result of this, A must choose action a2, rather than a1, which would have been optimal without agreements!

2.4 $h = 2 * h'$

Since h' multiplied the number of people by a large number, it does not matter if this large number is X or $2 * X$. Therefore, all of the results will remain the same, and the heuristic is still admissible.

Exercise 3 Game trees

3.1 Each agent out for itself, and they cannot communicate.

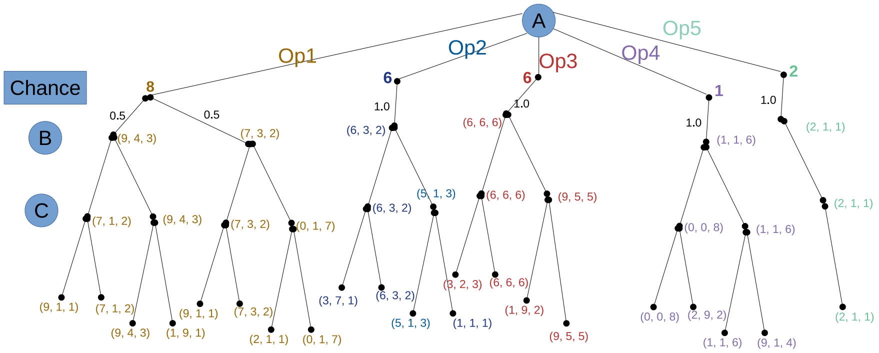


Figure 5: Option 1 is the best

3.2 Paranoid assumption: B and C are against A

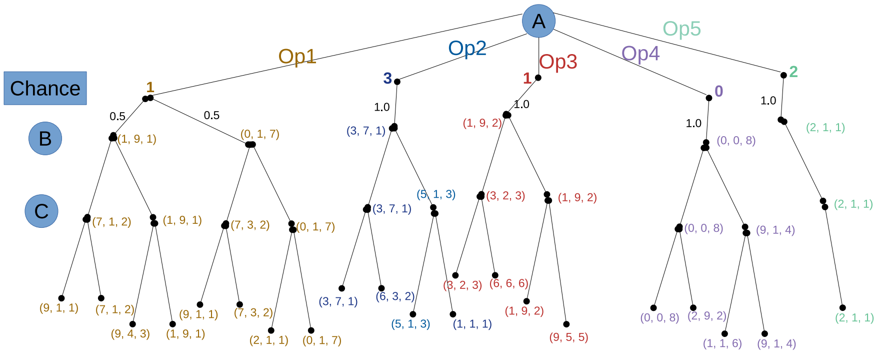


Figure 6: Option 2 is the best

3.3 B and C may agree on a deal if it is beneficial to both of them

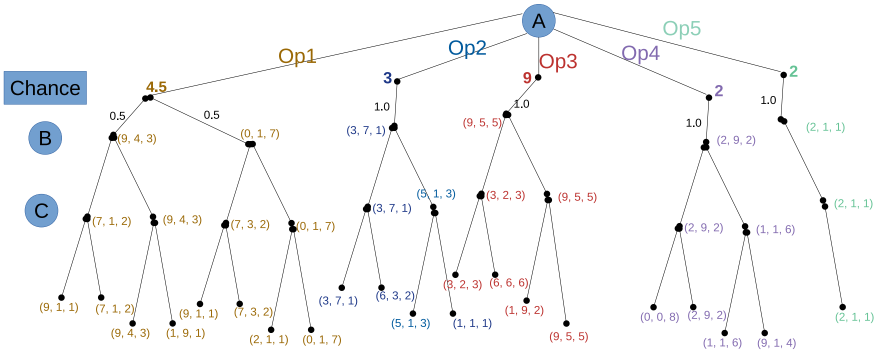


Figure 7: Option 3 is the best

3.4 A and C are partners aiming to maximize the sum of their scores

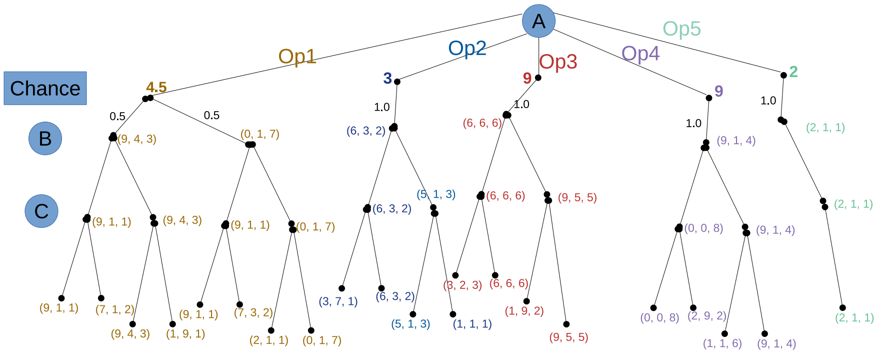


Figure 8: Option 4 is the best

3.5 A is a firm believer in Murphy's laws: "mother nature is a b*!=h", and wants toplay absolutely safe.

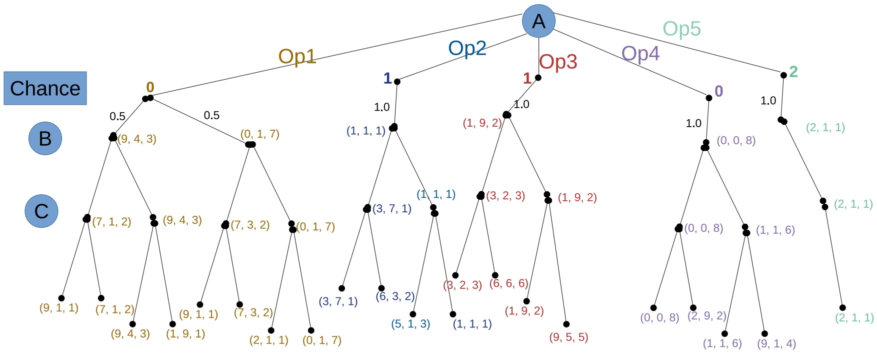


Figure 9: Option 5 is the best

Exercise 4 Game-tree search - alpha-beta pruning

4.1 Construct an example where the optimal first move is no-op

Assuming the world shown in Fig. 10. K=0 and Deadline=5. In this world B is a vandal agent that works as follows: He moves one turn, destroying the traversed edge, and then does NoOp() one turn. The agent A is at V4 and already carrying people (number does not matter). The agent wants to get to the shelter, however, he must see first what B will do. If B goes to V1, A have to be at V1 already, because next move B might go to the shelter and A won't be able to save the carrying people. However, if B goes to V2 or V3 first, the optimal solution will be to save an additional person.

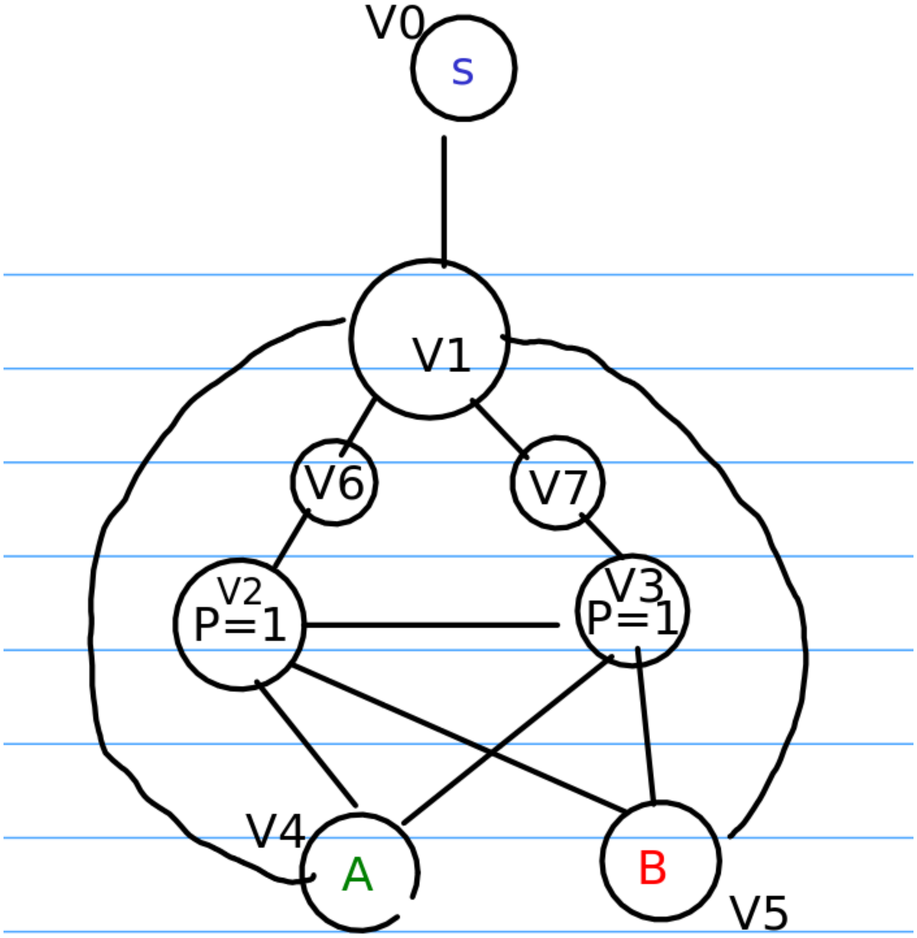


Figure 10: World where all edges are 1. B is a vandal agent that waits 1 turn in place, and one turn moves randomly and destroys the traversed edge. A is carrying 1 person and should wait to see what B does first.

4.2 Construct and show an example

Using the heuristic of calculating how many people will not be saved, the expansion tree is given for the choices of agent A. At first it can go to V2, V3 or do NoOp. For each option, the B agent will randomly go to either V2, V3 or V1 with probability of 1/3.

As seen by Fig. 11, If agent A will go to V2 or V3 there is a possibility that the agent will finish the game with a very high heuristic value, therefore, the h value is 466.

However, if agent A will do No-Op then it has enough time to react to agent's B decision.

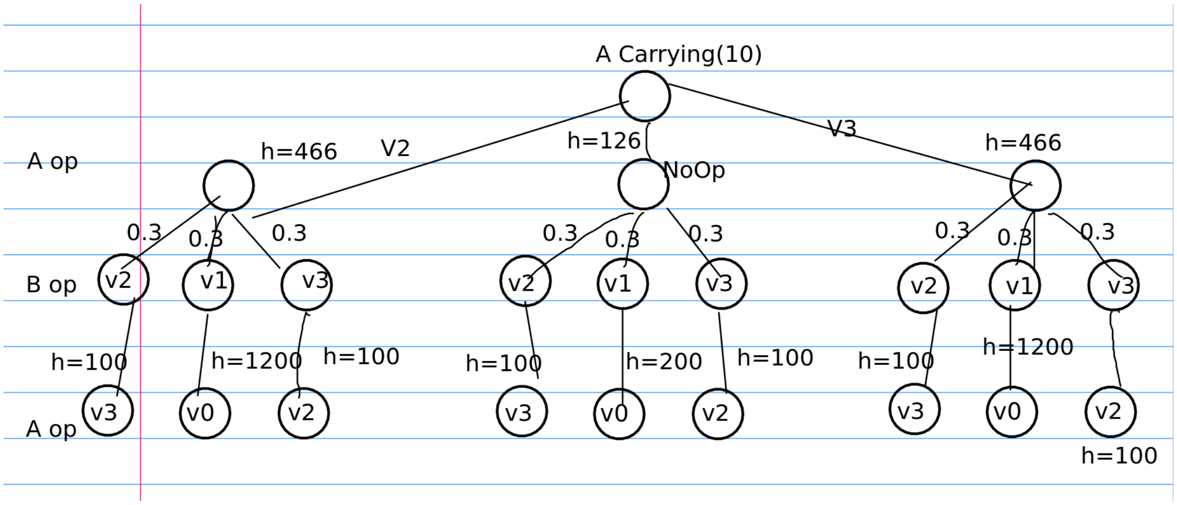


Figure 11: Expansion tree.

4.3 Show where alpha-beta pruning can decrease search effort in the tree from b

Assume that we start expanding V2. After the choice of agent B, several of the remaining choices can be discarded. In addition (and mostly), due to symmetry, expanding V3 will be futile and will finish this branch much faster.

Exercise 5 Propositional logic

For validity it will be shown that the sentence is always true.

An example for satisfiability a true model and a false model will be given.

For unsatisfiability it will be shown that the setence is always false.

Number of models equals to: 2^n, if the sentence is valid, where n is the number of symbols.

5.1 $(\neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge F) \vee (A \wedge B \wedge C \wedge D \wedge E)$

True model - A, B, C, D, F are true, therefore,

$True \vee False = True$

False model - A is true, B is false (the rest whatever), therefore,

$False \vee False = False$

Satisfiable. Number of satisfiable models is 3.

5.2 $(\neg A \lor \neg B \lor \neg C \lor \neg D \lor \neg E \lor \neg F) \land (A \lor B \lor C \lor D \lor E \lor F)$

False model - A, B, C, D, F are true, therefore,

$True \land False = False$

True model - A is true, B is false (the rest whatever), therefore,

$True \land True = True$

Satisfiable. Number of satisfiable models is 62.

5.3 $(A \lor B \land (D \lor \neg A) \land (E \lor A) \Rightarrow (B \lor C \land (\neg D \lor E))) \land (A \land \neg A)$

$\alpha \equiv (A \lor B \land (D \lor \neg A) \land (E \lor A) \Rightarrow (B \lor C \land (\neg D \lor E)))$

$\alpha \land (A \land \neg A) \to \alpha \land False \to False$

Unsatisfiable Number of models is 0.

5.4 $(A \land (A \Rightarrow B) \land (B \Rightarrow C)) \Rightarrow C$

implication elimination $\neg(A \land (\neg A \lor B) \land (\neg B \lor C)) \lor C$

Distributivity $\neg((A \land \neg A) \lor (A \land B)) \land (\neg B \lor C)) \lor C$

$\neg(False \lor (A \land B)) \land (\neg B \lor C)) \lor C$

Associativity $\neg(A \land (B \land (\neg B \lor C))) \lor C$

Distributivity $\neg(A \land ((B \land \neg B) \lor (B \land C))) \lor C$

$\neg(A \land ((False) \lor (B \land C))) \lor C$

$\neg(A \land (B \land C)) \lor C$

De Morgan $\neg A \lor \neg B \lor \neg C \lor C \to True$

Valid. Number of models is 8.

5.5 $\neg((A \land (A \Rightarrow B) \land (B \Rightarrow C)) \Rightarrow C)$

implication elimination $\neg(\neg(A \land (\neg A \lor B) \land (\neg B \land C)) \lor C)$

Distributivity $\neg(\neg((A \land \neg A) \lor (A \land B) \land (\neg B \land C)) \lor C)$

$\neg(\neg((A \land B) \land (\neg B \land C)) \lor C)$

De Morgan $\neg(\neg A \lor \neg B \lor \neg\neg B \lor \neg C \lor C)$

Double negation elimination $\neg(\neg A \lor \neg B \lor B \lor \neg C \lor C) \to$

$\neg(\neg A \lor True \lor True) \to$

$\neg(True) \to False$

Unsatisfiable. Number of models is 0.

Prep for exam 1

November 27, 2018

1 What we learned

- Search
  1. Single agent
  2. 0 - sum games (+stochastic)
- KB (logic)
  1. propositional
  2. FOL

2 Note on CNF and DNF

CNF: $(A \vee B) \wedge (\neg B \vee C)$

DNF $(x \wedge y) \vee (\neg x \wedge z)$

Transformation from DNF to CNF is NP-hard

3 Every bird has 2 wings mistake

$\forall x Bird(x) \Rightarrow [\exists w, z \quad Wing(w) \wedge Wing(z) \wedge (Has(x, w)) \wedge Has(x, z) \wedge \neg(w = z) \wedge \forall y (NOT \exists y)(wing(y) \wedge Has(x, y))] \Rightarrow y = w \vee y = z$

4 Exercise

$$\forall x \quad Syrian(x) \Rightarrow \exists y \quad Missile(y) \wedge Owns(x, y) \tag{1}$$

$$\exists x, y \quad Owns(Assad, y) \wedge Owns(Assad, x) \wedge M(x) \wedge M(y) \wedge \neg(x = y) \tag{2}$$

$$\forall x (Owns(Assad, x) \wedge M(x)) \Rightarrow Chemical(x) \tag{3}$$

$$\forall x \quad (\exists y \quad Chemical(y) \wedge Weapon(y) \wedge Owns(x, y)) \Rightarrow Evil(x) \tag{4}$$

$$\forall x \quad M(x) \Rightarrow W(x) \tag{5}$$

4.1 CNF:

Can remove EI since there are no negations. To remove EI need to make Skolem function (just say skolem function, don't define). In (1) its $MM1()$

$$\forall x \quad \neg Syrian(x) \vee Missile(MM1(x)) \wedge Owns(x, MM1(x))$$

Apply distributive rule

$$\forall x \quad \neg Syrian(x) \vee Missile(MM1(x)) \tag{6}$$

$$\forall x \quad \neg Syrian(x) \vee Owns(x, MM1(x)) \tag{7}$$

From (2) - remove 2 EI

$$O(A, M1) \tag{8}$$

$$O(A, M2) \tag{9}$$

$$M(M1) \tag{10}$$

$$M(M2) \tag{11}$$

$$\neg(M1 = M2) \tag{12}$$

From (3) we get (recommended not to use x but use x1 to avoid mistakes)

$$\neg O(A, x) \vee \neg M(x) \vee C(x) \tag{13}$$

From (4) we get

$$\forall x \quad \neg(\exists y \quad Chemical(y) \wedge Weapon(y) \wedge Owns(x, y)) \vee Evil(x)$$

$$\neg Chemical(y) \vee \neg Weapon(y) \vee \neg Owns(x, y)) \vee Evil(x) \tag{14}$$

From (5) we get

$$\neg M(x) \vee W(x) \tag{15}$$

4.2 Query:

Evil(Assad) Q'. : $\neg E(A)$

4.3 Resolution:

Resolve (13), (8), $\theta = (x/M1)$

$$\neg M(M1) \vee C(M1) \tag{16}$$

Resolve (16), (10), $\theta = ()$

$$C(M1) \tag{17}$$

Resolve (17), (14), $\theta = (y/M1)$

$$\neg W(M1) \vee \neg O(x, M1) \vee E(x) \tag{18}$$

Resolve (18), (8), $\theta = (x/A)$

$$\neg W(M1) \vee E(A) \tag{19}$$

Resolve (15), (10), $\theta = (x/M1)$

$$W(M1) \tag{20}$$

Resolve (19), (20), $\theta = ()$

$$E(A) \tag{21}$$

Resolve (21), Q', $\theta = (x/A)$

$$Square \tag{22}$$

4.4 Can use backward chaining?

Yes. All CNF are Horn clauses.

For Horn clause

- At INF should be only one implication.
- At CNF should be at most only one positive.

4.5 Can you prove Assad is Syrian?

$S(A)$
$\neg S(A)$

From the knowledge base it is impossible to prove that $\neg S(A)$ is true since there is no $S(A)$!

To prove S(A), resolution with (6) will give M(MM1(A)). It is impossible to continue from hereon. Therefore, can't prove that Assad is Syrian.

Knowledge base with contradiction resolves everything!

5 games - search

Play a two player game, non 0-sum. The game is in figure:

a.

How can I play to get best performance given that every player wants to maximize numbers.

Expand, and left branch is best: (50, 40)

Can you use branch trimming? No

b.

There is a 0.84 that the opponent plays randomly

6 True or False

$h(n)$ is more optimistic than $h(n)/5$? False

h(n) = -g(n) - is admissible? False. h(n) has to be 0 at the goal state!

Can you play Go with on a 19 x 19 board with a truth table? False. 3^308

$C \wedge B \Rightarrow A$, num of models is 7? True