Review-1

Decision Tree Solution

	P>=5	G>=100	BP>=65	ST>=25	I=0	BMI>25	DF>=10	A>=32	OutCome
0	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	1
1	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	0
2	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	1
3	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	0
4	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	1

Decision Tree Solution

OutCome	
0	3
1	2

$$Entropy(s) = -\frac{2}{5}log_2(\frac{2}{5}) - \frac{3}{5}log_2(\frac{3}{5})$$

$$Entropy(s) = 0.971$$

Decision Tree Solution

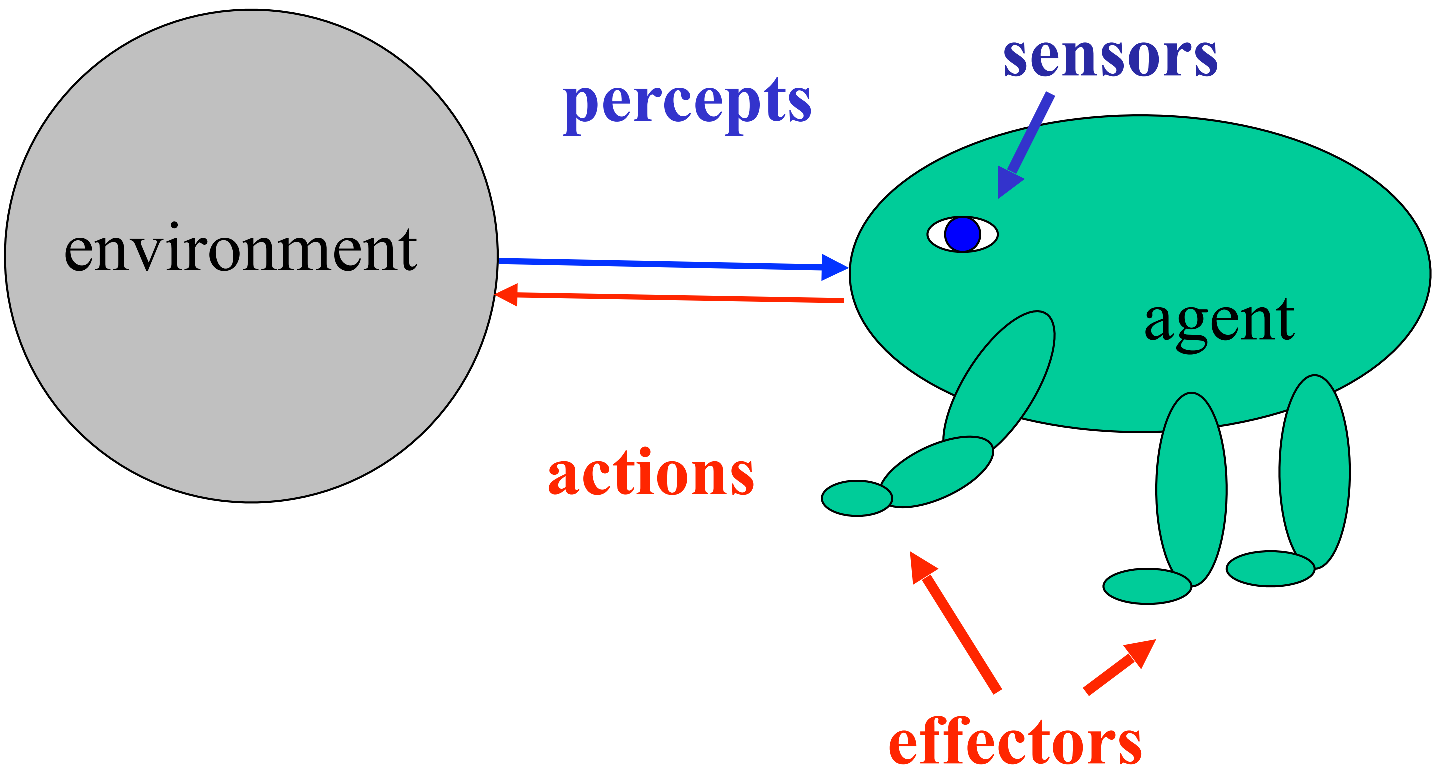
	OutCome		
G>=100		0	1
	TRUE	0	3
	FALSE	2	0

$$Entropy(G >= 100, OutCome) = P((G >= 100 : True)E(0,3) + P((G >= 100 : Flase)E(2,0)$$

$$Entropy(G >= 100, OutCome) = 0$$

$$Gain(G >= 100, OutCome) = Entropy(S) - Entropy(G >= 100, OutCome) = Entropy(S) - 0$$

Agent



PEAS Description

- Consider an "automated taxi driver"
  - **P**erformance Measure?
    - Safe, fast, obey laws, reach destination, comfortable trip, maximize profits
  - **E**nvironment?
    - Roads, other traffic, pedestrians, weather, customers
  - **A**ctuators?
    - Steering, accelerator, brake, signal, horn, speak, display
  - **S**ensors?
    - Cameras, microphone, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Basic Types of Agent Programs

- Simple reflex agents
  - Condition-action rules on <u>current</u> percept
  - Environment must be fully observable
- Model-based reflex agents
  - Maintain internal state about how world the world evolves and how actions effect the world
- Goal-based agents
  - Use goals and planning to help make decision
- Utility-based agents
  - What makes the agent “happiest”
- Learning agents
  - Makes improvements

Define Problems and Solutions

A problem is defined by four items

1. Initial state
2. Actions/Operators
3. Goal test
4. Path cost

A solution is a sequence of operators leading from the initial state to a goal state & Optimal solution has lowest path cost

Search Problems

- Uninformed Search
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

- Informed Search
  - Greedy search
  - A* Search

A Game Defined as Search Problem

- Initial state
  - Board position
  - Whose move it is
- Operators (successor function)
  - Defines legal moves and resulting states
- Terminal (goal) test
  - Determines when game is over (terminal states)
- Utility (objective, payoff) function
  - Gives numeric value for the game outcome at terminal states
  - e.g., {win = +1, loss = -1, draw = 0}

Adversarial Search

- In which we examine the problems that arise
  - when we try to plan ahead to get the best result
    - in a world that includes a <u>hostile</u> agent (other agent planning against us).

Minimax

- Perfect play for deterministic, perfect-information games
- Two players: MAX, MIN
  - MAX moves first, then take turns until game is over
  - Points are awarded to winner
- Choose move to position with highest *minimax* value

Alpha-beta pruning

- Ignore portions of search tree that make no difference to final choice
- Prunes away branches that <u>cannot possibly influence</u> final minimax decision
- Returns <u>same</u> move as general minimax

A Simple Knowledge-Based Agent

- Knowledge base saves:
  - Current state of world
  - How to infer unseen properties of world from percepts
  - How world evolves over time
  - What it wants to achieve
  - What its own actions do in various circumstances

The Language of KB

Syntax:

"$x + 2 \geq y$" is a sentence

"$x2 + y >$" is not a sentence

Semantics:

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is True in a world where x=7, y=1

$x + 2 \geq y$ is False in a world where x=0, y=6

Propositional Logic: Syntax

- *True*, *False*, S_1, S_2, … are sentences
- If S is a sentence, $\neg S$ is a sentence
  - Not (negation)
- $S_1 \wedge S_2$ is a sentence, also $(S_1 \wedge S_2)$
  - And (conjunction)
- $S_1 \vee S_2$ is a sentence
  - Or (disjunction)
- $S_1 \Rightarrow S_2$ is a sentence
  - Implies (conditional)
- $S_1 \Leftrightarrow S_2$ is a sentence
  - Equivalence (biconditional)

Propositional Logic: Semantics

- Semantics defines the rules for determining the truth of a sentence
  - (wrt a particular model)
- $\neg S$, is true iff S is false
- $S_1 \wedge S_2$, is true iff S_1 is true <u>and</u> S_2 is true
- $S_1 \vee S_2$, is true iff S_1 is true <u>or</u> S_2 is true
- $S_1 \Rightarrow S_2$, is true iff S_1 is false <u>or</u> S_2 is true
- $S_1 \Leftrightarrow S_2$, is true iff $S_1 \Rightarrow S_2$ is true <u>and</u> $S_2 \Rightarrow S_1$ is true
    - (S_1 same as S_2)

Propositional Inference: Enumeration Method

- Test $((P \vee H) \wedge \neg H) \Rightarrow P$

P	H	$P \vee H$	$\neg H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	False	False	True	False	True
False	True	True	False	False	True
True	False	True	True	True	True
True	True	True	False	False	True

Inference Rules for Prop. Logic

- Modus Ponens
  - From implication and premise of implication, can infer conclusion

$$\frac{\alpha \Rightarrow \beta, \ \ \alpha}{\beta}$$

- And-Elimination
  - From conjunction, can infer any of the conjuncts

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n}{\alpha_i}$$

Inference Rules for Prop. Logic

- And-Introduction
  - From list of sentences, can infer their conjunction

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n}$$

- Or-Introduction
  - From sentence, can infer its disjunction with anything else

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n}$$

Inference Rules for Prop. Logic

- Double-Negation Elimination
  - From doubly negated sentence, can infer a positive sentence

$$\frac{\neg\neg\alpha}{\alpha}$$

- Unit Resolution
  - From disjunction, if one of the disjuncts is false, can infer the other is true

$$\frac{\alpha \vee \beta, \quad \neg\beta}{\alpha}$$

- Resolution
  - β cannot be both true and false
  - One of the other disjuncts must be true in one of the premises (implication is transitive)

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Syntax of FOL: Basic Elements

- Constant symbols for specific objects
  KingJohn, 2, OSU, …
- Predicate (boolean) properties (unary) / relations (binary+)
  Brother(), Married(), >, …
- Functions (return objects)
  Sqrt() , LeftLegOf(), FatherOf(), …
- Variables
  x, y, a, b, …
- Connectives
  $\wedge \quad \vee \quad \neg \quad \Rightarrow \quad \Leftrightarrow$
- Equality
  $=$
- Quantifiers
  $\forall \quad \exists$

Quantifiers

- Currently have logic that allows objects
- Now want to express properties of entire <u>collections of objects</u>
  - Rather than enumerate the objects by name
- Two standard quantifiers
  - Universal $\forall$
  - Existential $\exists$

Quantifiers

Universal Qualification

- "For all …" (typically use implication $\Rightarrow$)
  - Allows for "rules" to be constructed
- $\forall$ *<variables> <sentence>*
  - Everyone at OSU is smart
    $\forall x \;\; At(x, OSU) \;\Rightarrow\; Smart(x)$

Existential Quantification

- "There exists …" (typically use conjunction $\wedge$)
  - Makes a statement about <u>some</u> object (not all)
- $\exists$ *<variables> <sentences>*
  - Someone at OSU is smart
    $\exists x \;\; At(x, OSU) \wedge Smart(x)$
- Uniqueness quantifier
  $\exists! \, x$ says a <u>unique</u> object exists (i.e. there is exactly one)

Properties of Quantifiers

- <u>Important</u> relations

$\exists x \;\; P(x) \;\; = \;\; \neg\forall x \;\; \neg P(x)$

$\forall x \;\; P(x) \;\; = \;\; \neg\exists x \, \neg P(x)$

$P(x) \Rightarrow Q(x)$ is same as $\neg P(x) \vee Q(x)$

$\neg \, (P(x) \wedge Q(x))$ is same as $\neg P(x) \vee \neg Q(x)$

Reduction to Propositional Inference

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)
  - $\exists x \; \forall y$
    - There must be some *x* for which the sentence is true with every possible *y*
  - $\forall x \; \exists y$
    - For every possible *x*, there must be some *y* that satisfies the sentence
    - Use a Skolem <u>function</u> instead

Skolem Function

- SK1(x) is effectively a function which returns a person that x loves.

- $\forall x\ \exists y$ *Skolem Substitution* Example

  1) $\forall x\ \exists y\ Person(x) \rightarrow Loves(x,y)$

  2) $\forall x\ Person(x) \rightarrow Loves(x, SK1(x))$ [Substitute, $\{y/SK1(x)\}$]

  3) $Person(Jack) \rightarrow Loves(Jack, SK1(Jack))$ [Then, $\{x/Jack\}$]

Reduction to Propositional Inference

- Internal Quantifiers should be moved outward
  - $\forall x\ (\exists y\ Loves(x,y)) \rightarrow Person(x)$
  - $\forall x\ \neg(\exists y\ Loves(x,y)) \vee Person(x)$ [convert to $\neg, \vee, \wedge$]
  - $\forall x\ \forall y\ \neg Loves(x,y) \vee Person(x)$ [move $\neg$ inward]
  - $\forall x\ \forall y\ Loves(x,y) \rightarrow Person(x)$

Forward Chaining

- Forward chaining normally triggered by addition of new fact to KB (using TELL)
- When new fact *p* added to KB:
  - For each rule such that *p* unifies with a premise
    - If the other premises are known, then add the conclusion to the KB and continue chaining

Forward Chaining: Example

Knowledge Base

$A \rightarrow B$

$A \rightarrow D$

$D \rightarrow C$

$A \rightarrow E$

$D \rightarrow F$

$E \rightarrow G$

Add A:

A, $A \rightarrow B$ gives B [done]

A, $A \rightarrow D$ gives D

D, $D \rightarrow C$ gives C [done]

D, $D \rightarrow F$ gives F [done]

A, $A \rightarrow E$ gives E

E, $E \rightarrow G$ gives G [done]

[done]

Order of generation B, D, C, F, E, G

Backward Chaining

- Backward chaining designed to find all answers to a question posed to KB (using ASK)
- When a query *q* is asked:
  - If a matching fact *q'* is known, return the unifier
  - For each rule whose consequent *q'* matches *q*
    - Attempt to prove each premise of the rule by backward chaining

Resolution

- Uses proof by contradiction
  - To prove *P*:
    - Assume *P* is FALSE
    - Add ¬*P* to KB
    - Prove a contradiction