STEVENS
INSTITUTE OF TECHNOLOGY
1870

Artificial Intelligence

Computer Science, CS541 - A

Jonggi Hong

Announcements

- HW #1 is due today (11:59 pm).

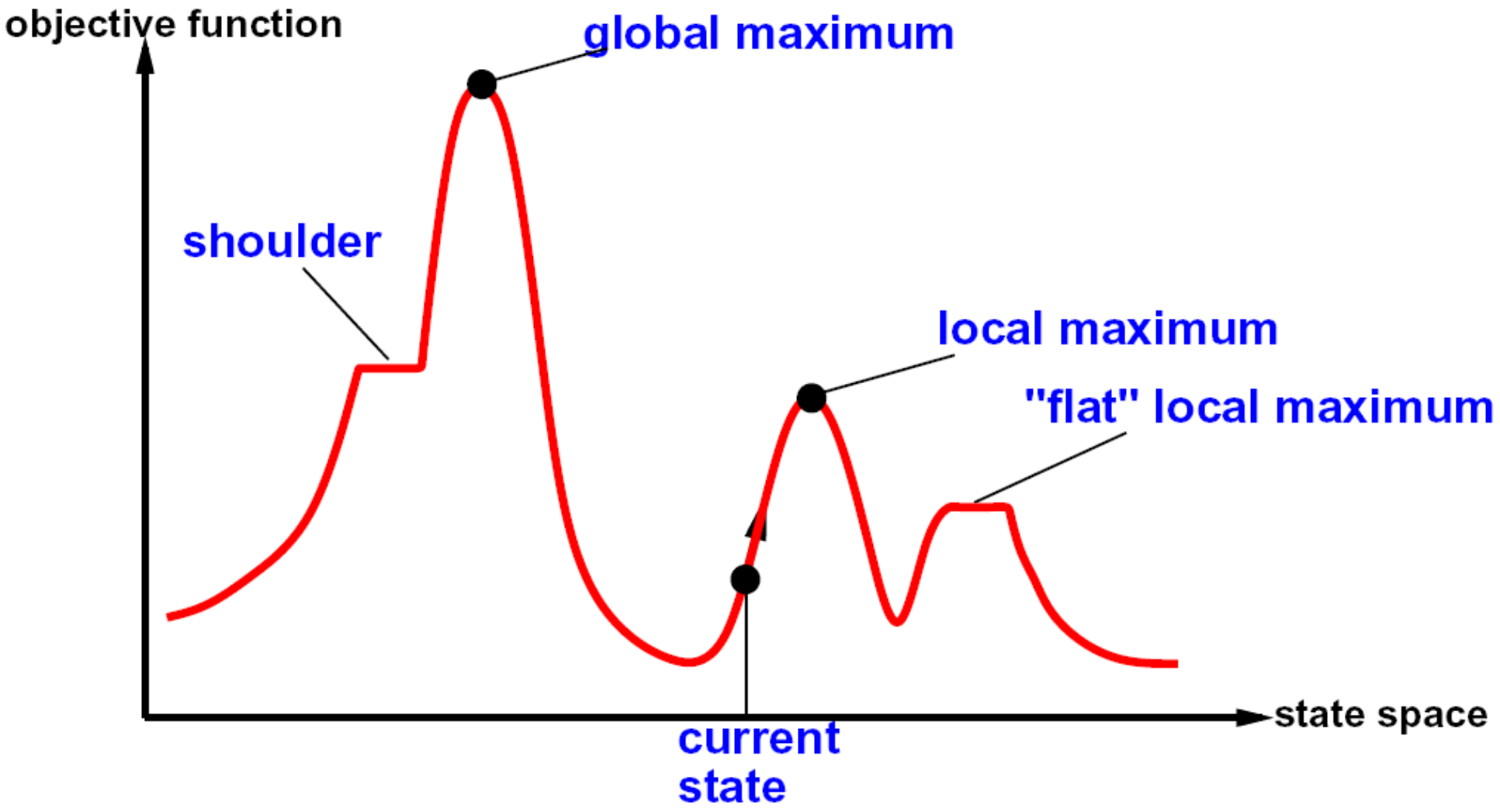
STEVENS
INSTITUTE OF TECHNOLOGY
1870

Recap

Local search and adversarial search

Local Search

Hill climbing diagram



Simulated annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

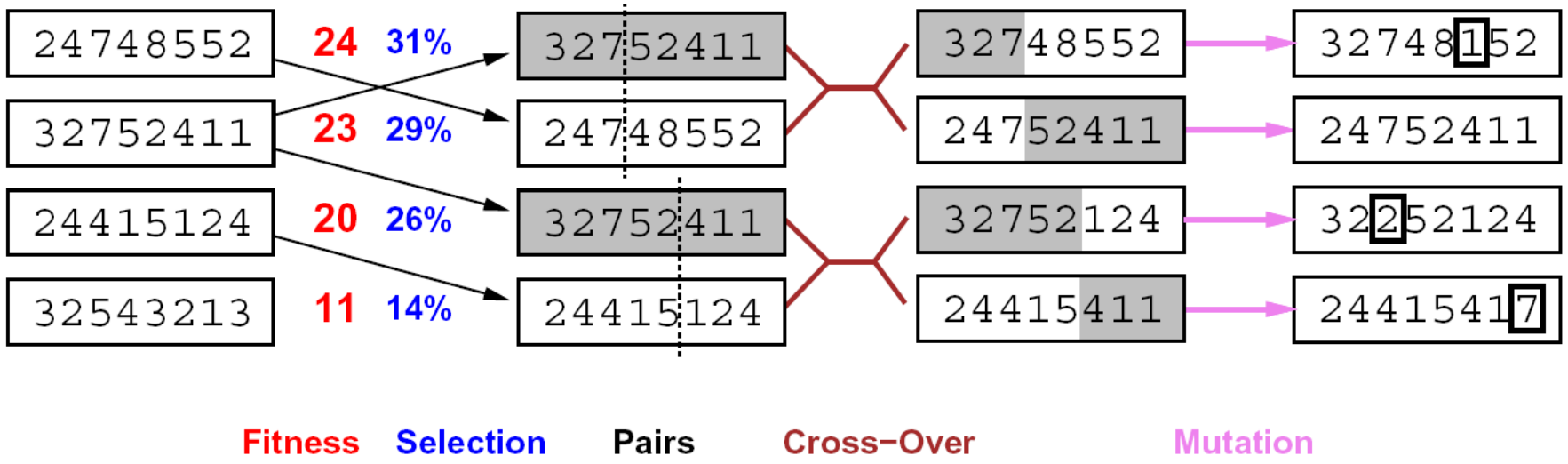
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward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

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

Typical genetic algorithm

1. Initialize: Population P←N random individuals
2. Evaluate: For each x in P, compute fitness(x)
3. Loop
   - For i=1 to N
     - **Select** 2 parents each with probability proportional to fitness scores
     - **Crossover** the 2 parents to produce a new samples (child)
     - With some small probability **mutate** child
     - Add child to population
   - Until some child is fit enough or you get bored
4. Return best child in the population according to fitness function

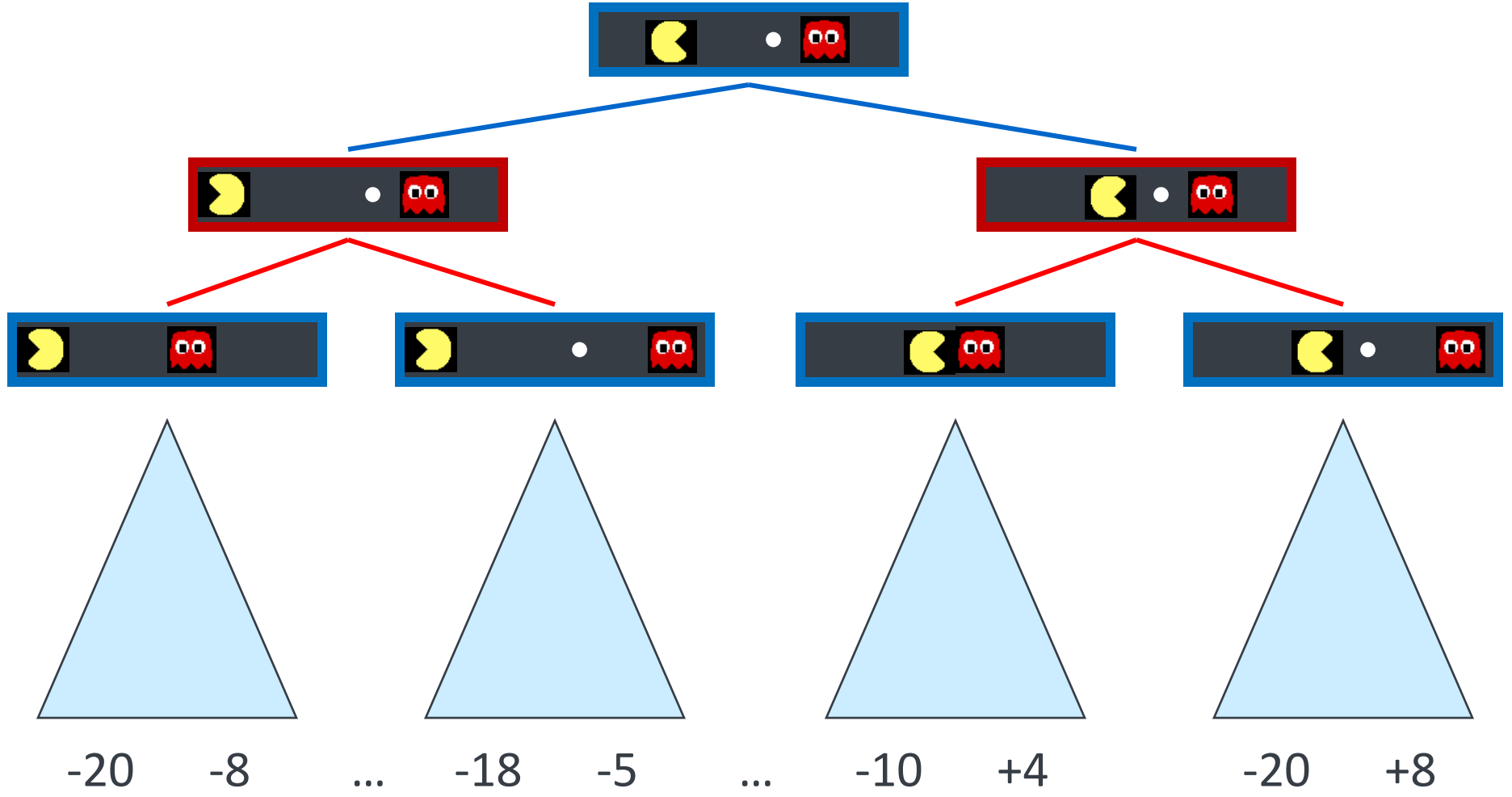
Genetic algorithms



- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Adversarial Search

Adversarial game trees



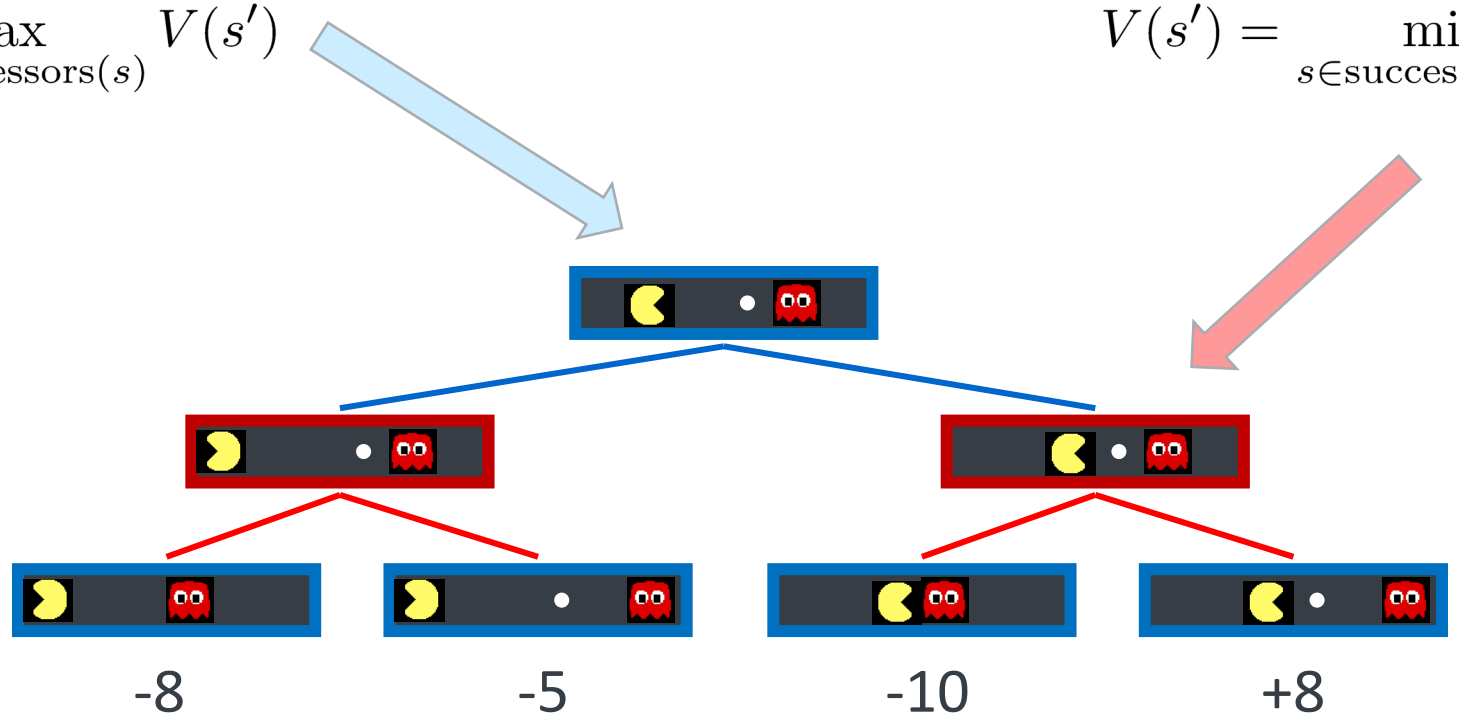
Minimax values

States under agent's control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States under opponent's control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

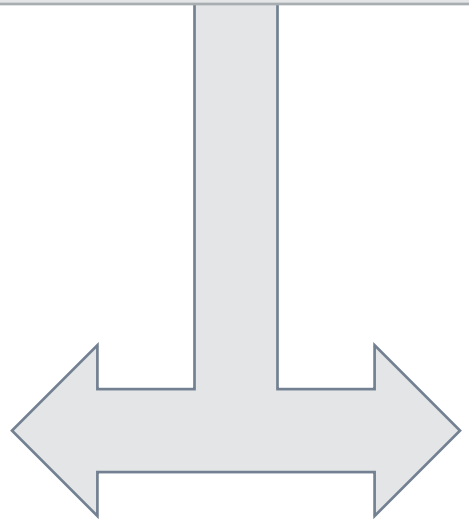
$$V(s) = \text{known}$$

Minimax implementation (dispatch)

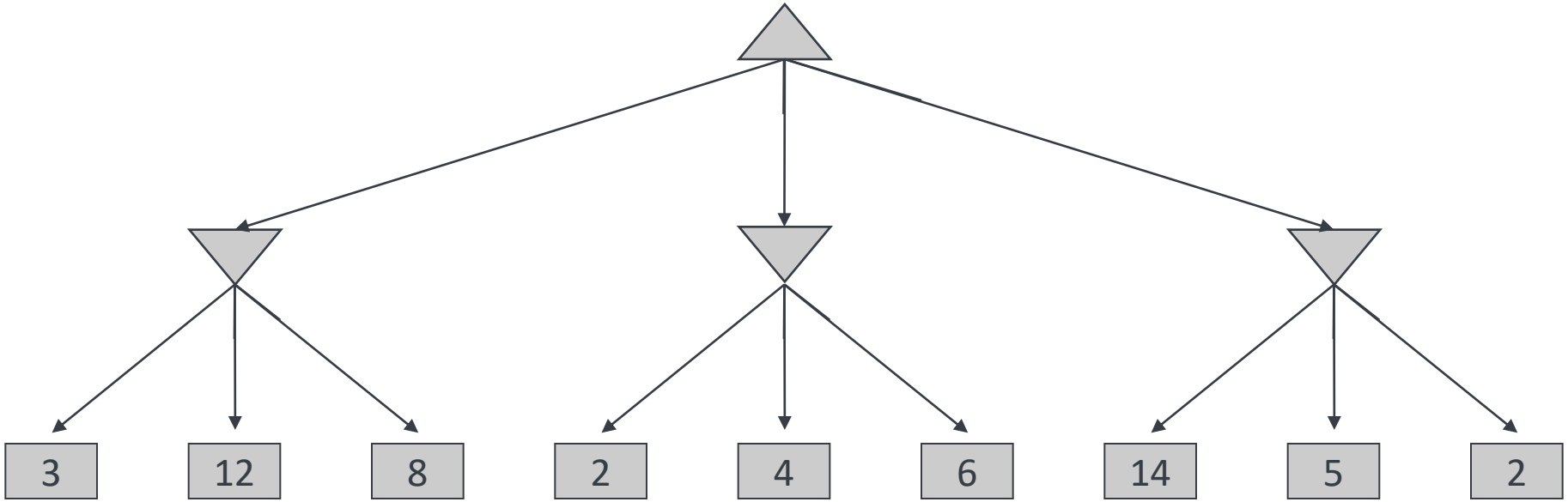
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

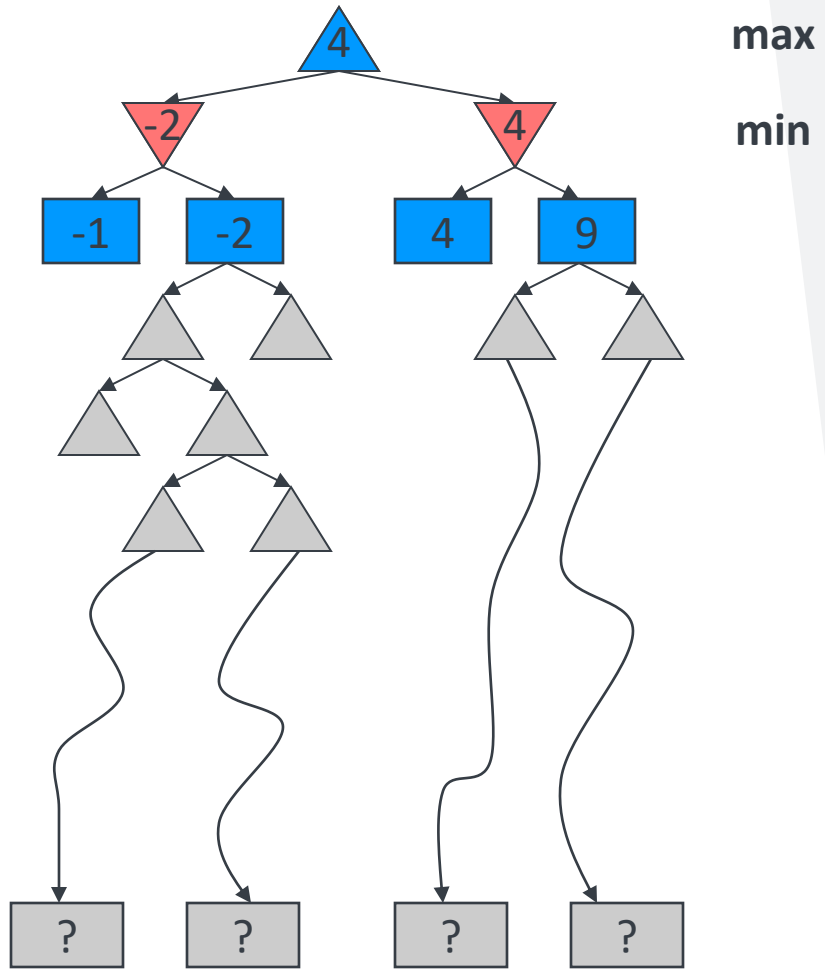


Minimax example



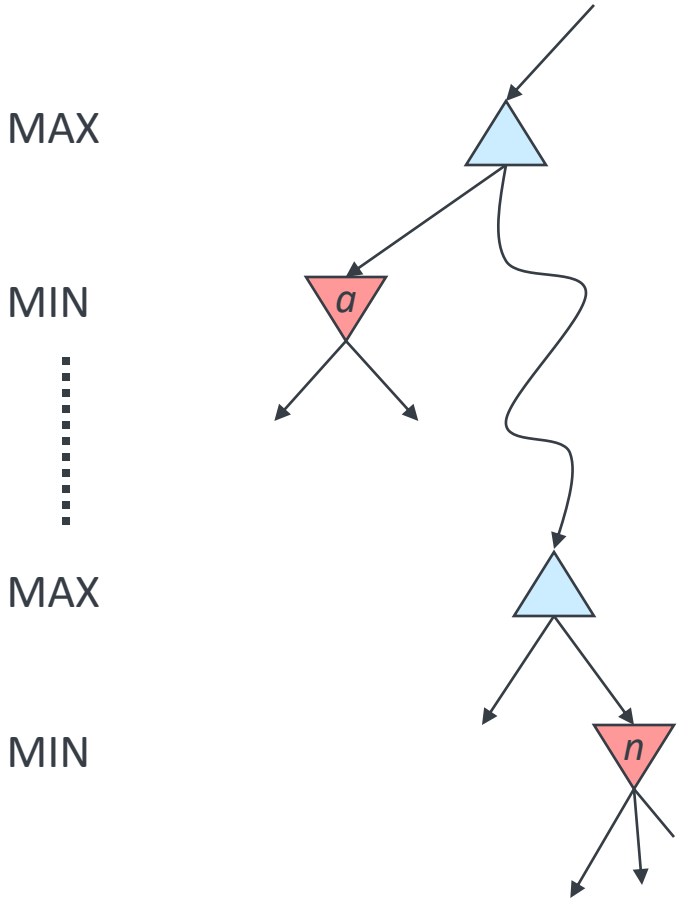
Resource limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an **evaluation function** for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



Alpha-beta pruning

- 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 children's min is dropping
  - Who cares about *n*'s value? MAX
  - Let *α* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *α*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



- MAX version is symmetric

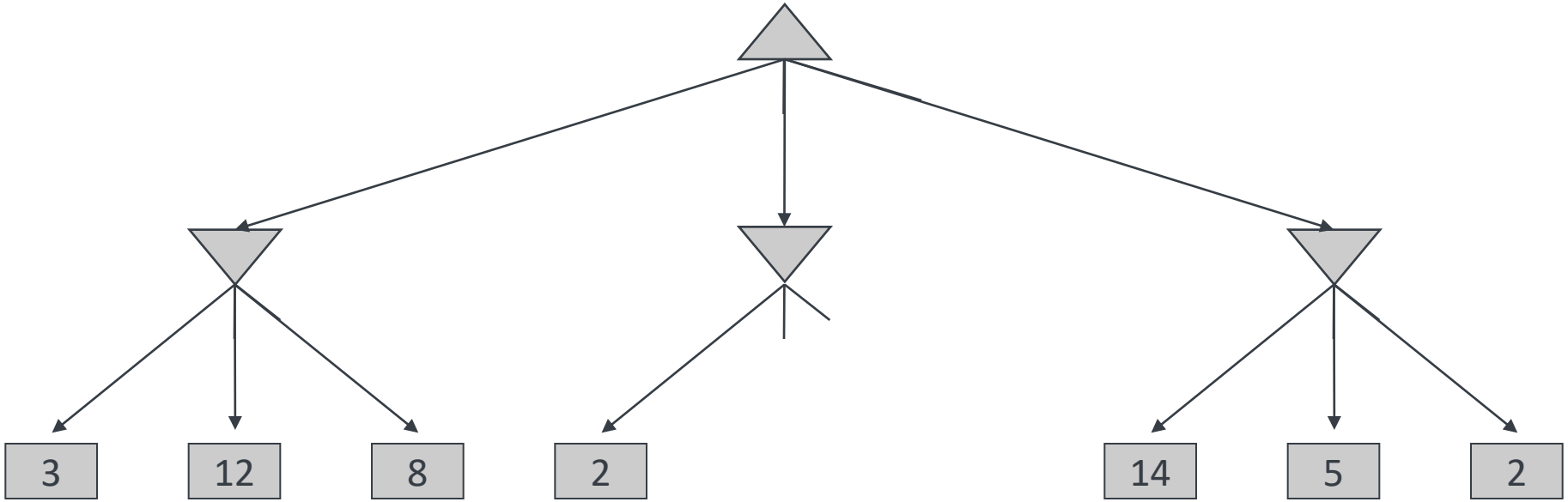
Alpha-beta implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

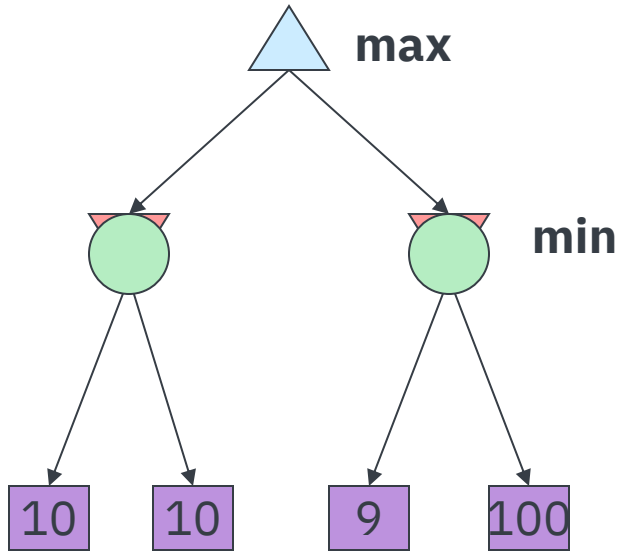
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Minimax pruning



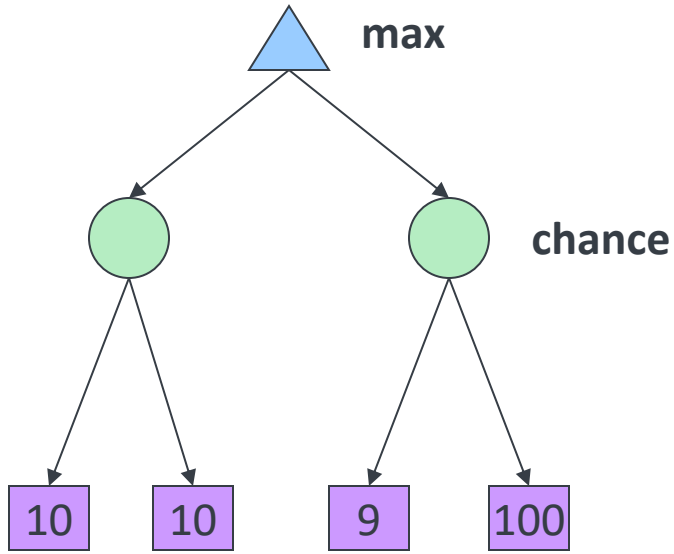
Worst-case vs. Average case



Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- **Expectimax search**: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their **expected utilities**
  - *I.e.* take weighted average (expectation) of children



Expectimax pseudocode

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Adversarial Search (cont.)

Chapter 5

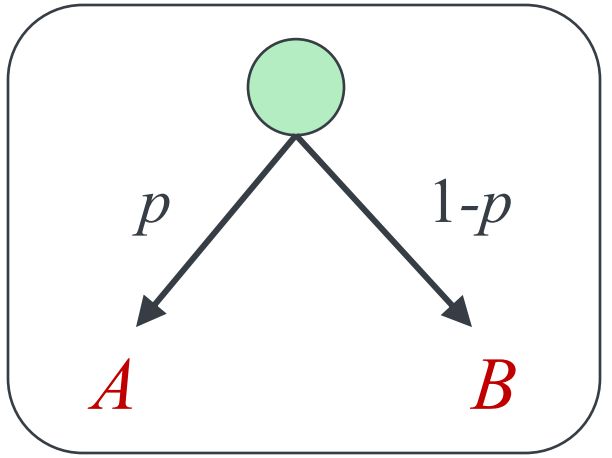
Preferences

- An agent must have preferences among:
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes

$$L = [p, A;\ \ (1-p), B]$$

A Prize

A Lottery



- Notation:
  - Preference: $A \succ B$
  - Indifference: $A \sim B$

Rationality

Rational preferences

- We want some constraints on preferences before we call them rational, such as:

Axiom of Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

- For example: an agent with **intransitive preferences** can be induced to give away all of its money
  - If B > C, then an agent with C would pay (say) 1 cent to get B
  - If A > B, then an agent with B would pay (say) 1 cent to get A
  - If C > A, then an agent with A would pay (say) 1 cent to get C

Rational preferences

The axioms of rationality

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

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

Theorem: Rational preferences imply behavior describable as maximization of expected utility.

MEU principle

- Theorem [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \;\Leftrightarrow\; A \succeq B$$

$$U([p_1, S_1;\; \ldots\; ;\; p_n, S_n]) = \sum_i p_i U(S_i)$$

  - *I.e.* values assigned by U preserve preferences of both prizes and lotteries!

- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner

Human utilities

Utility scales

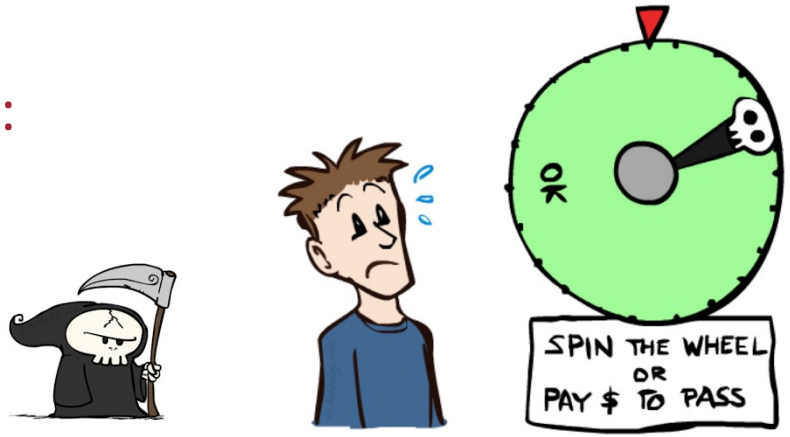
- **Normalized utilities**: $u_+ = 1.0$, $u_- = 0.0$
- **Micromorts**: one-millionth chance of death, useful for paying to reduce product risks, etc.
- **QALYs**: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

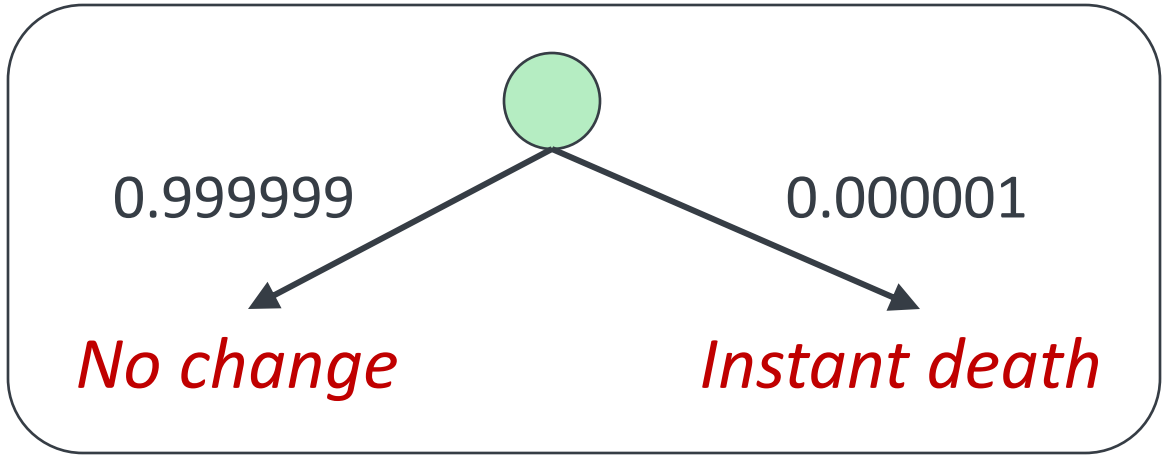
- With deterministic prizes only (no lottery choices), only ordinal utility can be determined, *i.e.*, total order on prizes

Human utilities

- Utilities map states to real numbers. Which numbers?
- Standard approach to assessment (elicitation) of human utilities:
  - Compare a prize A to a standard lottery L_p between
    - "best possible prize" u_+ with probability p
    - "worst possible catastrophe" u_- with probability 1-p
  - Adjust lottery probability p until indifference: A ~ L_p
  - Resulting p is a utility in [0,1]

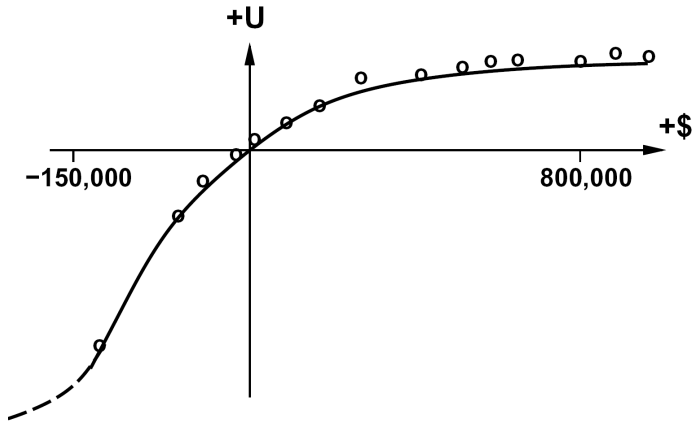


Pay $30 ~



Money

- Money does not behave as a utility function, but we can talk about the utility of having money (or being in debt)
- Given a lottery L = [p, $X; (1-p), $Y]
  - The expected monetary value EMV(L) is p*X + (1-p)*Y
  - U(L) = p*U($X) + (1-p)*U($Y)
  - Typically, U(L) < U(EMV(L))
  - In this sense, people are risk-averse
  - When deep in debt, people are risk-prone



Example: Insurance

- Consider the lottery [0.5, $1000; 0.5, $0]
  - What is its expected monetary value? ($500)
  - What is its certainty equivalent?
    - Monetary value acceptable in lieu of lottery
    - $400 for most people
  - Difference of $100 is the insurance premium
    - There's an insurance industry because people will pay to reduce their risk
    - If everyone were risk-neutral, no insurance needed!
  - It's win-win: you'd rather have the $400 and the insurance company would rather have the lottery (their utility curve is flat and they have many lotteries)

Which one do you prefer? - 1

A: [0.8, $4k; 0.2, $0]

B: [1.0, $3k; 0.0, $0]

Which one do you prefer? -2

C: [0.2, $4k; 0.8, $0]

D: [0.25, $3k; 0.75, $0]

Example: human rationality?

- Famous example of Allais (1953)
  - A: [0.8, $4k; 0.2, $0]
  - B: [1.0, $3k; 0.0, $0]
  - C: [0.2, $4k; 0.8, $0]
  - D: [0.25, $3k; 0.75, $0]
- Most people prefer B > A, C > D
- But if U($0) = 0, then
  - B > A $\Rightarrow$ U($3k) > 0.8 U($4k)
  - C > D $\Rightarrow$ 0.8 U($4k) > U($3k)

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Logic

Chapter 5

Language

- Natural language (informal)
  - English: Two divides even numbers.
  - German: Zwei dividieren geraden zaheln.
- Programming language (formal)
  - Python: def even(x): return x % 2 == 0
  - C++: bool even(int x) { return x % 2 == 0; }
- Logical languages (formal)
  - First-order logic: ∀x. Even(x) → Divides(x, 2)

Two Goals of a Logic Language

- Represent knowledge about the world
- Reason with that knowledge

What is Logic?

- A formal language
  - **Syntax**: what expressions are legal
  - **Semantics**: what legal expressions mean
  - **Proof system**: a way of manipulating syntactic expressions to get other syntactic expressions (which will tell us something new)
- Why proofs?
  - Two kinds of inferences an agent might want to make:
  - Multiple percepts → conclusions about the world
  - Current state & operator → properties of the next state

What is Logic?

- Propositional logic
- First-order logic

Propositional Logic

Propositional Logic Syntax

- Syntax: what you are allowed to write
  - for (thing t=fizz; t==fuzz; t++) { … }
  - “Colorless green ideas sleep furiously.”
- Sentences (wffs: well formed formulas)
  - true and false are sentences.
  - Propositional variables are sentences: P, Q, R, Z
  - If φ and ψ are sentences, then so are
    - (φ), ¬ φ, φ ∨ ψ, φ ∧ ψ, φ → ψ, φ ↔ ψ
  - Nothing else is a sentence.

Precedence

highest ¬

∧

∨

→

lowest ↔

$A \vee B \wedge C$	$A \vee (B \wedge C)$
$A \wedge B \rightarrow C \vee D$	$(A \wedge B) \rightarrow (C \vee D)$
$A \rightarrow B \vee C \leftrightarrow D$	$(A \rightarrow (B \vee C)) \leftrightarrow D$

- Precedence rules enable “shorthand” form of sentences, but formally only the fully parenthesized form is legal.
- Syntactically ambiguous forms allowed in shorthand only when semantically equivalent: $A \wedge B \wedge C$ is equivalent to $(A \wedge B) \wedge C$ and $A \wedge (B \wedge C)$.

Semantics

- Meaning of a sentence is truth value {**t**, **f**}
- **Interpretation (model)** is an assignment of truth values to the propositional variables.
  - holds(φ, i) [Sentence φ is **t** in interpretation i]
  - fails(φ, i) [Sentence φ is **f** in interpretation i]

Semantic Rules

- holds(true, i) for all i
- fails(false, i) for all i
- holds(¬φ, i) if and only if fails fails(φ, i) — negation
- holds(φ ∧ ψ, i) iff holds(φ, i) and holds(ψ, i) — conjunction
- holds(φ ∨ ψ, i) iff holds(φ, i) or holds(ψ, i) — disjunction
- holds(P, i) iff i(P) = **t**
- fails(P, i) iff i(P) = **f**

Some Important Shorthand

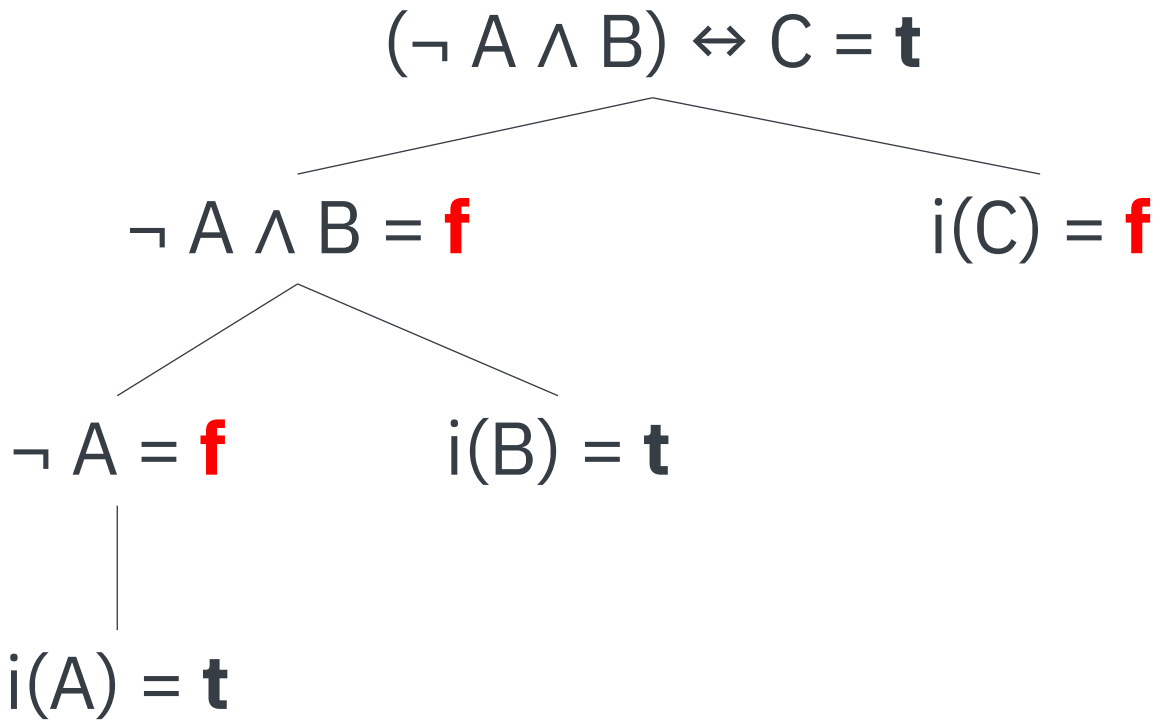
- $\varphi \rightarrow \psi \equiv \neg \varphi \vee \psi$ (conditional, implication)
  - Implication: antecedent → consequent
- $\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$ (biconditional, equivalence)

P	Q	¬ P	P ∧ Q	P ∨ Q	P → Q	Q → P	P ↔ Q
f	**f**	**t**	**f**	**f**	**t**	**t**	**t**
f	**t**	**t**	**f**	**t**	**t**	**f**	**f**
t	**f**	**f**	**f**	**t**	**f**	**t**	**f**
t	**t**	**f**	**t**	**t**	**t**	**t**	**t**

Note that implication is not “causality”, if P is **f** then P→ Q is **t**.

Example: Inverted V-Structure

- Sentence
  - $(\neg A \wedge B) \leftrightarrow C$
- Interpretation
  - i(A) = **t,** i(B) = **t,** i(C) = **f**



Terminology

- A sentence is **valid** iff its truth value is **t** in all interpretations.
  - Valid sentences: true, ¬ false, P ∨ ¬ P
- A sentence is **satisfiable** iff its truth value is **t** in at least one interpretation.
  - Satisfiable sentences: P, true, ¬ P
- A sentence is **unsatisfiable** iff its truth value is **f** in all interpretations.
  - Unsatisfiable sentences: P ∧ ¬ P, false, ¬ true

These are finitely decidable in propositional logic.

Examples

Sentence	Valid?	Interpretation that make sentence's truth value = **f**
smoke → smoke	yes	
smoke ∨ ¬ smoke	yes	
smoke → fire	satisfiable, not valid	smoke = **t**, fire = **f**
(s → f) → (¬ s → ¬ f)	satisfiable, not valid	s = **f**, f = **t** s → f = **t**, ¬ s → ¬ f = **f**
(s → f) → (¬ f → ¬ s) (contrapositive)	valid	
b ∨ d ∨ (b → d)	valid	
b ∨ d ∨ ¬ b ∨ d	valid	

Satisfiability

- Related to constraint satisfaction
- Given a sentence S, try to find an interpretation i such that holds(S, i).
- Analogous to find an assignment of values to variables such that the constraints hold.

- Brute force method
  - enumerate all interpretations and check
- Better methods
  - heuristic search
  - constraint propagation
  - stochastic search

Satisfiability Problems

- Scheduling nurses to work in a hospital
  - Propositional variables represent the availability of nurses. (For example, Pat is working on Tuesday at 2.)
  - Constraints on the schedule are represented using logical expressions over the variables.
- Finding bugs in software
  - Propositional variables represent the state of the program.
  - Use logic to describe how the program works and to assert there is a bug.
  - If the sentence is satisfiable, you've found a bug!

A Good Lecture?

- Imagine we knew that
  - If today is sunny, then Tomas will be happy. (S → H)
  - If Tomas is happy, the lecture will be good. (H → G)
  - Today is sunny. (S)

- Should we conclude that the lecture will be good?

Checking Interpretations

S	H	G
t	t	t
t	t	f
t	f	t
t	f	f
f	t	t
f	t	f
f	f	f
f	f	f

Checking Interpretations

S	H	G	S → H	H → G	S
t	t	t	t	t	t
t	t	f	t	f	t
t	f	t	f	t	t
t	f	f	f	t	t
f	t	t	t	t	f
f	t	f	t	f	f
f	f	f	t	t	f
f	f	f	t	t	f

Knowledge base (KB)

Checking Interpretations

S	H	G	S → H	H → G	S	G
t	t	t	t	t	t	t
t	t	f	t	f	t	f
t	f	t	f	t	t	t
t	f	f	f	t	t	f
f	t	t	t	t	f	t
f	t	f	t	f	f	f
f	f	f	t	t	f	t
f	f	f	t	t	f	f

Good lecture!

Adding a Variable

Another variable: Leslie is happy. (L)

L	S	H	G	S → H	H → G	S	G
t	t	t	t	t	t	t	t
t	t	t	f	t	f	t	f
t	t	f	t	f	t	t	t
t	t	f	f	f	t	t	f
t	f	t	t	t	t	f	t
t	f	t	f	t	f	f	f
t	f	f	f	t	t	f	t
t	f	f	f	t	t	f	f
f	t	t	t	t	t	t	t
f	t	t	f	t	f	t	f
…	…	…	…	…	…	…	…

If we know that $A \vee B$ and $\neg B \vee C$ are true, what do we know about $A \vee C$?

- $A \vee C$ is guaranteed to be true
- $A \vee C$ is guaranteed to be false
- We don't have enough information to say anything definitive about $A \vee C$

Interpretations

If we know that $A \vee B$ and $\neg B \vee C$ are true, what do we know about $A \vee C$?

A	B	C	A v B	¬B v C	A v C
t	t	t	t	t	t
t	t	f	t	f	t
t	f	t	t	t	t
t	f	f	t	t	t
f	t	t	t	t	t
f	t	f	t	f	f
f	f	t	f	t	t
f	f	f	f	t	f

A v C is guaranteed to be true.

Interpretations

If we know that $A \vee B$ and $\neg B \vee C$ are true, what do we know about A?

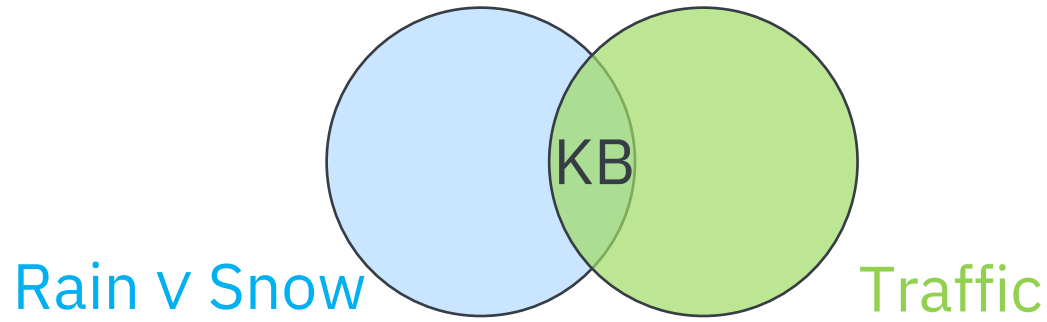
A	B	C	A ∨ B	¬B ∨ C	A ∨ C
t	t	t	t	t	t
t	t	f	t	f	t
t	f	t	t	t	t
t	f	f	t	t	t
f	t	t	t	t	t
f	t	f	t	f	f
f	f	t	f	t	t
f	f	f	f	t	f

We don't have enough information to say anything definitive about A.

Knowledge Base

- A **knowledge base** (KB) is a set of sentences representing their conjunction/intersection.
- Intuition
  - KB specifies constraints on the world. KB represents the set of interpretations that satisfies those constraints.

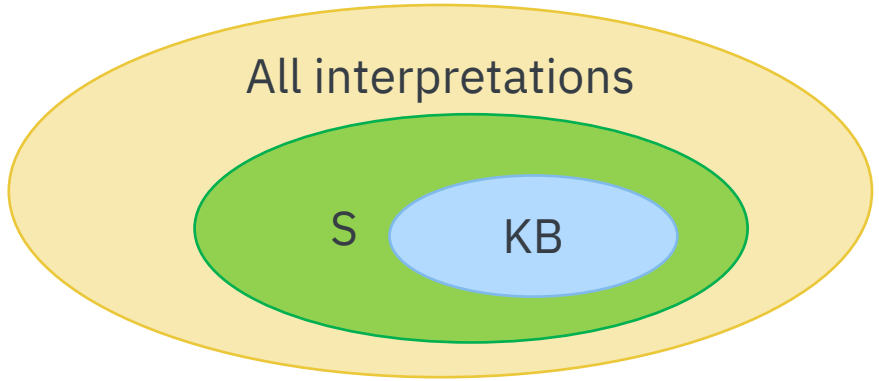
Let KB = {Rain v Snow, Traffic}



Entailment

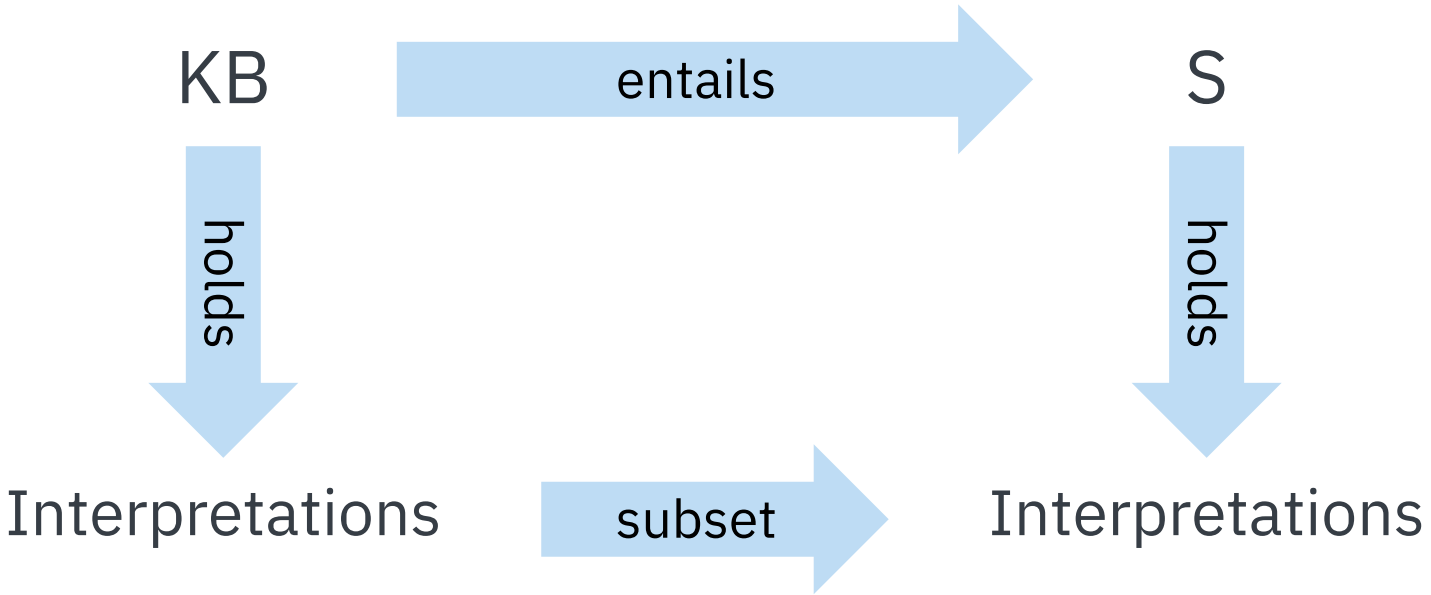
A knowledge base (KB) **entails** a sentence S iff every interpretation that makes KB true also makes S true.

- Intuition
  - The sentence adds no information/constraints (it was already known)
- Entailment
  - KB entails S (written KB $\models$ S)
- Example
  - Rain $\wedge$ Snow $\models$ Snow



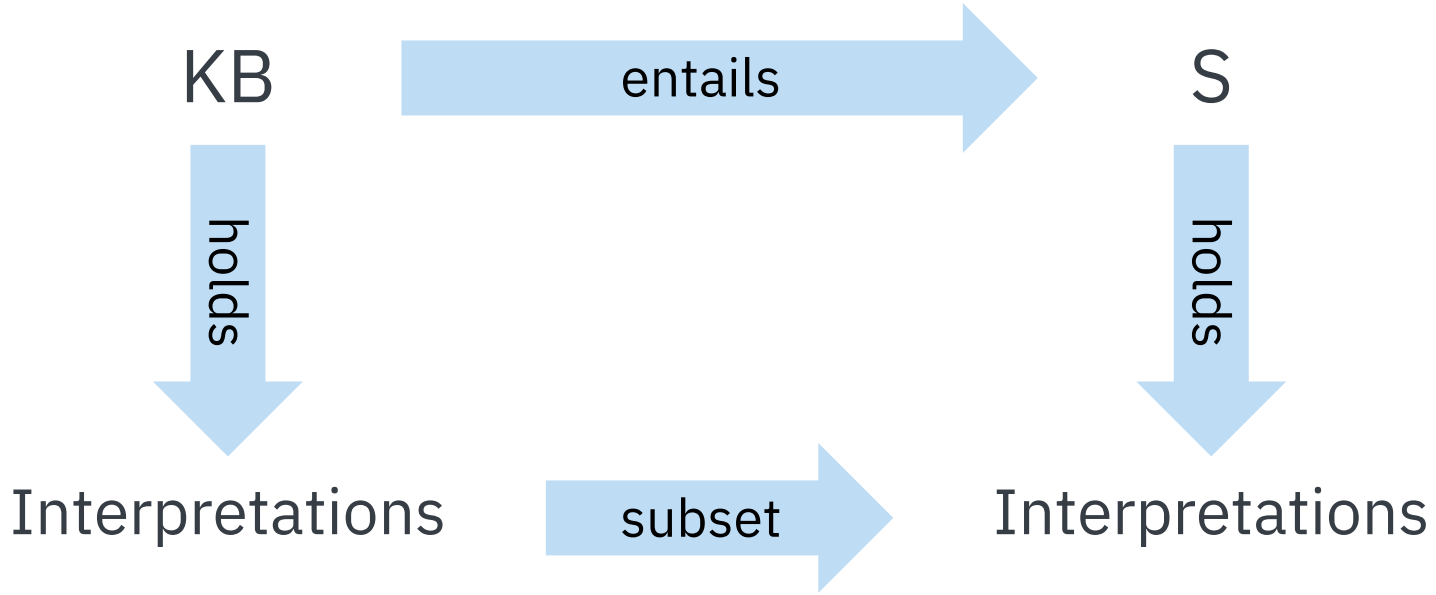
Entailment

A knowledge base (KB) **entails** a sentence S iff every interpretation that makes KB true also makes S true.



Computing Entailment

1. Enumerate all interpretations.
2. Select those in which all elements of KB are true.
3. Check to see if S is true in all of those interpretations.

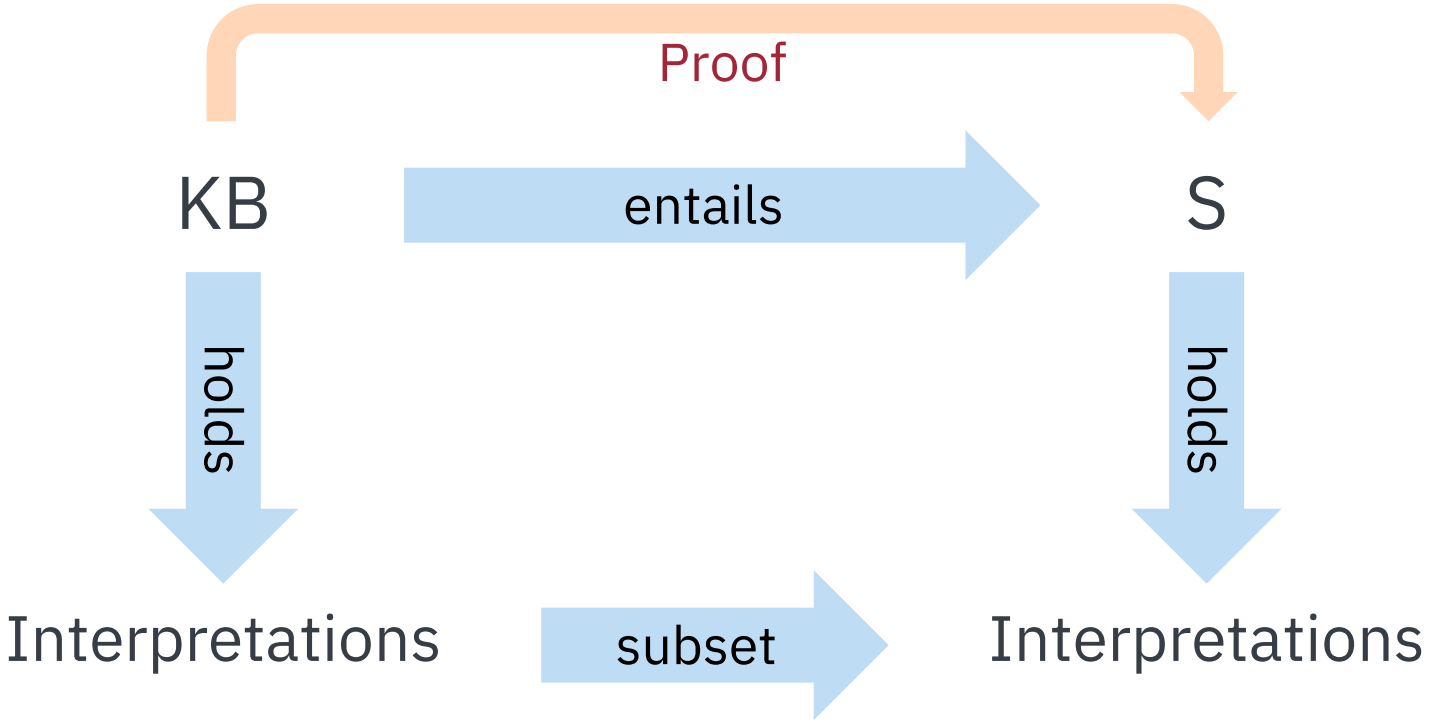


Usually, there are too many interpretations.

2^{10} =1024 interpretations with 10 variables.

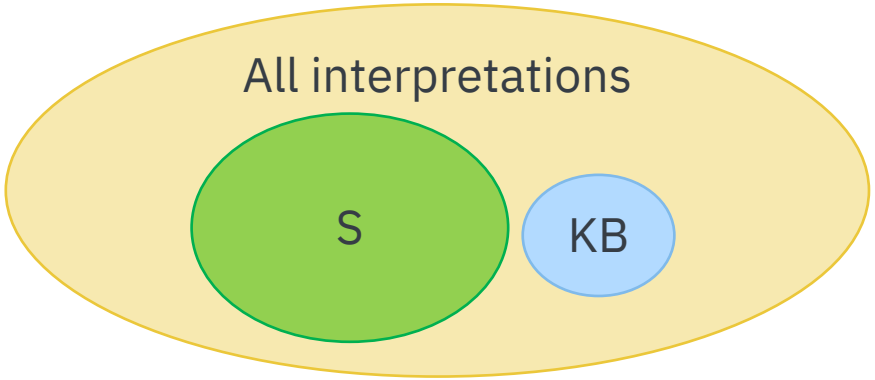
Entailment and Proof

A **proof** is a way to test whether a KB entails a sentence, without enumerating all possible interpretations.

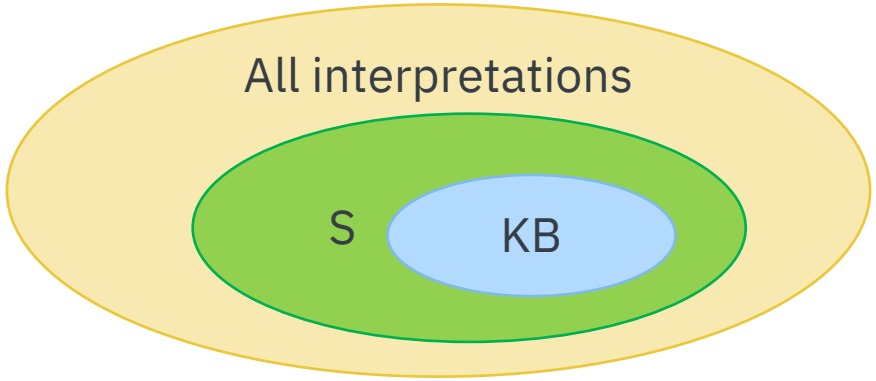


Contradiction

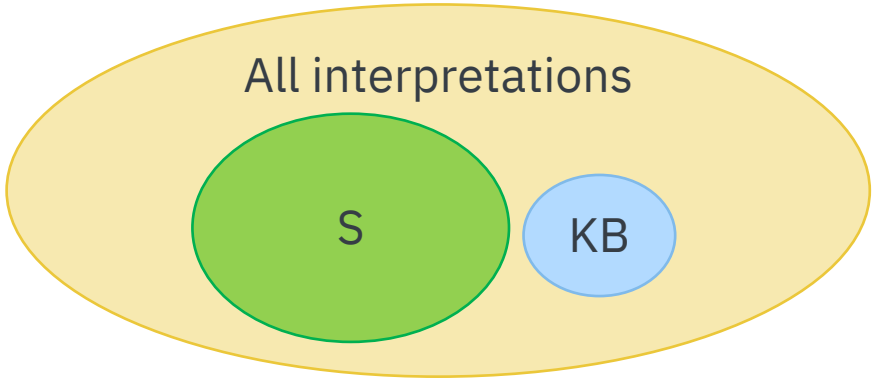
- Intuition
  - The sentence contradicts what we know (captured in KB)
- Contradiction
  - KB contradicts f iff $M(\text{KB}) \cap M(\text{S}) = \emptyset$
- Example
  - Rain ∧ Snow contradicts ¬ Snow



Entailment and Contradiction



Entailment



Contradiction

KB contradicts S iff KB entails ¬S.

Proof

- Proof is a sequence of sentences.
- First ones are premises (KB).
- Then, you can write down on the next line the result of applying an inference rule to previous lines.
- When S is on a line, you have proved S from KB.

Proof

- Example of making an inference
  - It is raining. (Rain)
  - If it is raining, then it is wet. (Rain → Wet)
  - Therefore, it is wet. (Wet)

$$\frac{\text{Rain} \qquad \text{Rain} \rightarrow \text{Wet}}{\text{Wet}}$$

(Modus ponens rule)

Entailment and Proof

- Semantics
  - Interpretation defines entailed/true sentences
  - KB $\models$ S (entailment)

- Syntax
  - Inference rules derive sentences
  - KB $\vdash$ S (proof)

Proof

- If inference rules are **sound**, then any S you can prove from KB is entailed by KB.
  - $KB \vdash S \rightarrow KB \models S$

- If inference rules are **complete**, then any S that is entailed by KB can be proved from KB.
  - $KB \models S \rightarrow KB \vdash S$

Soundness and Completeness

- Knowledge base
  - Rain
  - Rain → Wet
- Sentence
  - Wet
- Inference rule
  - $\left\{ \dfrac{\alpha \rightarrow \beta \quad \alpha}{\beta} \right\}$

Is the inference rule sound?

yes

Is the inference rule complete?

yes

Soundness and Completeness

- Knowledge base
  - Rain
  - Rain v Snow → Wet
- Sentence
  - Wet
- Inference rule
  - $\left\{ \dfrac{\alpha \rightarrow \beta \quad \alpha}{\beta} \right\}$

Is the inference rule sound?

yes

Is the inference rule complete?

no

- Semantically: KB ⊨ S
- Syntactically: KB ⊭ S

Fixing Completeness

- Option 1
  - Restrict the allowed set of sentences.
  - Propositional logic → propositional logic with only Horn clauses.

- Option 2
  - Use more powerful inference rules

STEVENS
INSTITUTE OF TECHNOLOGY
1870

15 min. break

Natural Deduction

Natural Deduction

- Example of making an inference
  - It is raining. (Rain)
  - If it is raining, then it is wet. (Rain → Wet)
  - Therefore, it is wet. (Wet)

$$\frac{\text{Rain} \qquad \text{Rain} \rightarrow \text{Wet}}{\text{Wet}}$$

(Modus ponens rule)

Natural Deduction

- Some inference rules

$$\frac{\alpha \rightarrow \beta \quad \alpha}{\beta}$$

Modus ponens

$$\frac{\alpha \rightarrow \beta \quad \neg \beta}{\neg \alpha}$$

Modus tolens

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta}$$

And-introduction

$$\frac{\alpha \wedge \beta}{\alpha}$$

And-elimination

Natural Deduction Example

Step	Formula	Derivation
1	$P \land Q$	Given
2	$P \rightarrow R$	Given
3	$(Q \land R) \rightarrow \mathbf{S}$	Given

Natural Deduction Example

Step	Formula	Derivation
1	$P \land Q$	Given
2	$P \to R$	Given
3	$(Q \land R) \to S$	Given
4	P	1 And-Elim

Natural Deduction Example

Step	Formula	Derivation
1	$P \land Q$	Given
2	$P \rightarrow R$	Given
3	$(Q \land R) \rightarrow S$	Given
4	P	1 And-Elim
5	R	4, 2 Modus Ponens

Natural Deduction Example

Step	Formula	Derivation
1	$P \land Q$	Given
2	$P \rightarrow R$	Given
3	$(Q \land R) \rightarrow S$	Given
4	P	1 And-Elim
5	R	4, 2 Modus Ponens
6	Q	1 And-Elim

Natural Deduction Example

Step	Formula	Derivation
1	$P \land Q$	Given
2	$P \rightarrow R$	Given
3	$(Q \land R) \rightarrow S$	Given
4	P	1 And-Elim
5	R	4, 2 Modus Ponens
6	Q	1 And-Elim
7	$Q \land R$	5, 6 And-Intro

Natural Deduction Example

Step	Formula	Derivation
1	$P \land Q$	Given
2	$P \rightarrow R$	Given
3	$(Q \land R) \rightarrow S$	Given
4	P	1 And-Elim
5	R	4, 2 Modus Ponens
6	Q	1 And-Elim
7	$Q \land R$	5, 6 And-Intro
8	**S**	7, 3 Modus Ponens

Natural Deduction Algorithm

Input: set of inference rules Rules

Repeat until no changes to KB:

Choose a set of formulas f_1, …, f_k in KB

If matching rule $\frac{f1, \ldots, fk}{S}$ exists:

Add S to KB

KB derives/proves S (KB $\vdash$ S) iff S eventually gets added to KB.

Proof Systems

- There are many natural deduction (ND) systems.
  - They are typically “proof checkers”.
  - The ND systems can be complete and sound with propositional logic.
- Natural deduction uses lots of inference rules which introduces a large branching factor in the search for a proof.
- In general, you need to do “proof by cases” which introduces even more branching.

Prove R

1	P ∨ Q
2	Q → R
3	P → R

Propositional Resolution

Definite Clauses

- A **definite clause**

  $(p_1 \wedge p_2 \wedge \ldots \wedge p_k) \rightarrow q$

  where $p_1 \wedge p_2 \wedge \ldots \wedge p_k$, q are positive propositional symbols
- Intuition

  If hold, $p_1 \wedge p_2 \wedge \ldots \wedge p_k$ then q holds.
- Examples of definite clauses
  - (Rain ∧ Snow) → Traffic
  - Traffic
- Examples of non-definite clauses
  - ¬ Traffic
  - (Rain ∧ Snow) → (Traffic ∨ Peaceful)

Horn Clauses

- A **Horn clause** is either
  - a definite clause: $(p_1 \wedge p_2 \wedge \ldots \wedge p_k) \rightarrow q$
  - a goal clause: $(p_1 \wedge p_2 \wedge \ldots \wedge p_k) \rightarrow \mathbf{f}$
- Examples
  - (Rain ∧ Snow) → Traffic (definite)
  - Traffic ∧ Accident → **f** (goal)
    - equivalent: ¬ (Traffic ∧ Accident)

Completeness of Modus Ponens

$$\frac{\alpha \rightarrow \beta \quad \alpha}{\beta}$$

Modus ponens

- Claim

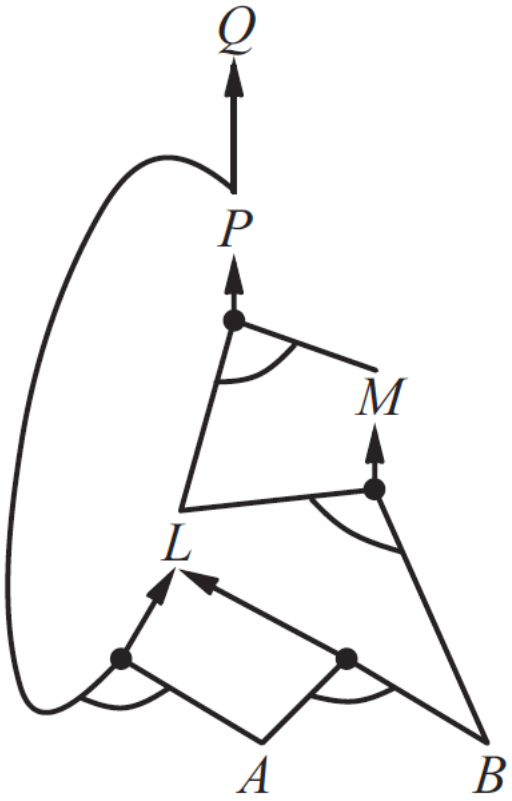
  Modus ponens is complete with respect to Horn clauses.

  - Suppose KB contains only Horn clauses and *p* is an entailed propositional symbol.
  - Then applying modus ponens will derive *p*.
  - $KB \models S \rightarrow KB \vdash S$

Forward Chaining

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Set of Horn clauses



AND-OR graph

- links with arc: conjunction
- links without arc: disjunction

Forward Chaining

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*
  inputs: KB, the knowledge base, a set of propositional definite clauses
    q, the query, a proposition symbol
  $count \leftarrow$ a table, where $count[c]$ is the number of symbols in c's premise
  $inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
  $agenda \leftarrow$ a queue of symbols, initially symbols known to be true in KB

  while $agenda$ is not empty **do**
    $p \leftarrow$ POP($agenda$)
    if $p = q$ **then return** *true*
    if $inferred[p] = false$ **then**
      $inferred[p] \leftarrow true$
      for each clause c in KB where p is in c.PREMISE **do**
        decrement $count[c]$
        if $count[c] = 0$ **then** add c.CONCLUSION to $agenda$
  return *false*

Horn Clauses and Disjunction

- Horn clauses can be converted to disjunctive clauses

Horn clauses	Disjunctive clauses
$A \rightarrow C$	$\neg A \vee C$
$A \wedge B \rightarrow C$	$\neg A \vee \neg B \vee C$

- Literal
  - Either p or ¬p, where p is a propositional symbol
- Clause
  - Disjunction of literals

Propositional Resolution

- Resolution rule

$$\frac{\alpha \vee \beta \qquad \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

- A single inference rule is a sound and complete proof system.
- Requires all sentences to be converted to conjunctive normal form.

Conjunctive Normal Form

- Conjunctive normal form (CNF) formulas

Example: $(A \vee B \vee \neg C) \wedge (B \vee D) \wedge (\neg A) \wedge (B \vee C)$

$(A \vee B \vee \neg C)$ is a **clause.**

A clause is a disjunction of literals.

A, B, ¬ C are **literals**.

A literal is a variable or a negation of a variable.

- Each clause is a requirement that must be satisfied and can be satisfied in multiple ways.
- Every sentence in propositional logic can be written in CNF.

Converting to CNF

1. Eliminate arrows (implications) using definitions.
2. Drive in negations using De Morgan's Laws.

$$\neg (\varphi \vee \psi) \equiv \neg \varphi \wedge \neg \psi$$
$$\neg (\varphi \wedge \psi) \equiv \neg \varphi \vee \neg \psi$$

3. Distribute **or** over **and.**

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

Every sentence can be converted to CNF, but it may grow exponentially in size.

CNF Conversion Example

$$(A \vee B) \rightarrow (C \rightarrow D)$$

1. Eliminate arrows (implications) using definitions.

$$\neg (A \vee B) \vee (\neg C \vee D)$$

2. Drive in negations using De Morgan's Laws.

$$(\neg A \wedge \neg B) \vee (\neg C \vee D)$$

3. Distribute **or** over **and.**

$$(\neg A \vee \neg C \vee D) \wedge (\neg B \vee \neg C \vee D)$$

Propositional Resolution

- Resolution rule

$$\frac{\alpha \vee \beta \qquad \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

- Resolution refutation
  - Convert all sentences to CNF.
  - Negate the desired conclusion (converted to CNF).
  - Apply resolution rule until either
    - Derive false (a contradiction)
    - Can’t apply any more

Propositional Resolution

- Resolution refutation is sound and complete.
  - If we derive a contradiction, then the conclusion follows from the axioms.
  - If we can’t apply any more, then the conclusion cannot be proved from the axioms.

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion
5	Q ∨ R	1, 2

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion
5	Q ∨ R	1, 2
6	¬ P	2, 4

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion
5	Q ∨ R	1, 2
6	¬ P	2, 4
7	¬ Q	3, 4

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion
5	Q ∨ R	1, 2
6	¬ P	2, 4
7	¬ Q	3, 4
8	R	5, 7

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

$$\frac{\text{false} \vee R \qquad \neg R \vee \text{false}}{\text{false} \vee \text{false}}$$

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion
5	Q ∨ R	1, 2
6	¬ P	2, 4
7	¬ Q	3, 4
8	R	5, 7
9	•	4, 8

Propositional Resolution Example

Prove R

1	P ∨ Q
2	Q → R
3	P → R

false ∨ R

¬ R ∨ false

---

false ∨ false

Step	Formula	Derivation
1	P ∨ Q	Given
2	¬ P ∨ R	Given
3	¬ Q ∨ R	Given
4	¬ R	Negated conclusion
5	Q ∨ R	1, 2
6	¬ P	2, 4
7	¬ Q	3, 4
8	R	5, 7
9	•	4, 8

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation
1	P	Given
2	¬ P	Given
3	¬ Z	Negated conclusion

The Power of False

Prove Z

1	P
2	¬ P

Step	Formula	Derivation
1	P	Given
2	¬ P	Given
3	¬ Z	Negated conclusion
4	•	1, 2

- Note that (P ∧ ¬ P) → Z is valid.
- Any conclusion follows from a contradiction – and so strict logic systems are very brittle.

Propositional Resolution Example 2

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Propositional Resolution Example 2

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Convert to CNF

$\neg (\neg P \vee Q) \vee Q$
$(P \wedge \neg Q) \vee Q$
$(P \vee Q) \wedge (\neg Q \vee Q)$
$(P \vee Q)$

$\neg (\neg P \vee P) \vee R$
$(P \wedge \neg P) \vee R$
$(P \vee R) \wedge (\neg P \vee R)$

$\neg (\neg R \vee S) \vee \neg (\neg S \vee Q)$
$(R \wedge \neg S) \vee (S \wedge \neg Q)$
$(R \vee S) \wedge (\neg S \vee S) \wedge (R \vee \neg Q) \wedge (\neg S \vee \neg Q)$
$(R \vee S) \wedge (R \vee \neg Q) \wedge (\neg S \vee \neg Q)$

Propositional Resolution Example 2

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Step	Formula	Derivation
1	$P \vee Q$	Given
2	$P \vee R$	Given
3	$\neg P \vee R$	Given
4	$R \vee S$	Given
5	$R \vee \neg Q$	Given
6	$\neg S \vee \neg Q$	Given
7	$\neg R$	Negated conclusion

Propositional Resolution Example 2

Prove R

1	$(P \rightarrow Q) \rightarrow Q$
2	$(P \rightarrow P) \rightarrow R$
3	$(R \rightarrow S) \rightarrow \neg(S \rightarrow Q)$

Step	Formula	Derivation
1	$P \vee Q$	Given
2	$P \vee R$	Given
3	$\neg P \vee R$	Given
4	$R \vee S$	Given
5	$R \vee \neg Q$	Given
6	$\neg S \vee \neg Q$	Given
7	$\neg R$	Negated conclusion
8	S	4, 7
9	$\neg Q$	6, 8
10	P	1, 9
11	R	3, 10
12	•	7, 11

Proof Strategies

- Unit preference

  Prefer a resolution step **involving a unit clause** (clause with one literal)

  - It produces a shorter clause. Our goal is to produce a zero-length clause!

- Set of support

  Choose a resolution **involving the negated goal** or any clause derived **from the negated goal**.

  - We are trying to produce a contradiction that follows from the negated goal. These are relevant clauses.
  - If a contradiction exists, it can be found with the set-of-support strategy.

First-Order Logic

Limitations of Propositional Logic

- Propositional logic only deals with fact statements that may or may not be true of the world.
  - Ex) R = “It is raining”
  - You cannot have variables that stands for books or tables.

- Examples
  - When you paint a block with green paint, it becomes green.
    - You would need a statement about every single block. You cannot make a general statement about all blocks.
  - When you sterilize a jar, all the bacteria are dead.
    - In First-Order Logic, we can talk about all bacteria without naming them explicitly.

First-Order Logic (FOL)

- **Variables** refer to things in the world.
- You can **quantify** over the variables.
  - You can make sentences talking about all of them or some of them without having to name them explicitly.

FOL Syntax

- Term
  - **Constant symbols**
    - Fred, Japan, Bacterium39
  - **Variables**
    - x, y, a
  - **Function symbol applied to one or more terms**
    - f(x), f(f(x)), mother-of(John)

FOL Syntax

- Sentence
  - A predicate symbol applied to zero or more terms
    - On(a, b), Sister(Jane, Joan), Sister(mother-of(John), Jane)
  - $t_1 = t_2$
  - If v is a variable and Φ is a sentence, then ∀v.Φ and ∃v.Φ are sentences.
  - Closure under sentential operators: ∨ ∧ ¬ → ↔ ( )

FOL Interpretations

- Interpretation I
  - U set of objects
    - called “domain of discourse” or “universe”
  - Maps constant symbols to elements of U
  - Maps predicate symbols to relations on U
    - An n-ary relation is a set of lists of n objects. A binary relation is a set of pairs.
  - Maps function symbols to functions on U
    - A function takes one or more arguments and returns a value. A predicate takes one or more arguments and is either true or false.
    - Assume our functions are *total* (*i.e.*, a function that returns a well-defined value for every element of its domain, with no undefined or "partial" values).

Denotation of Terms

- Terms name objects in U
  - I(Fred)
    - If Fred is constant, then given.
  - I(x)
    - If x is a variable, then undefined.
  - $I(f(t_1, ..., t_n))$
    - $I(f)(I(t_1), ..., I(t_n))$

Holds

- When does a sentence hold in an interpretation?
  - P is a relation symbol.
  - t_1, …, t_n are terms.

$$\text{holds}(P(t_1, \ldots, t_n), I) \text{ iff } \langle I(t_1), \ldots, I(t_n)\rangle \in I(P)$$

Brother(Jon, Joe)?

- I(Jon) = 😀 [an element of U]
- I(Joe) = 😎 [an element of U]
- I(Brother) = {<😁,😞>, < 😀, 😎 >, <…, …>, …}

Equality

- When are two terms equal?

$\text{holds}(t_1 = t_2, I)$ iff $I(t_1)$ is the same object as $I(t_2)$.

Jon = Jack?

- I(Jon) = 😀 [an element of U]
- I(Jack) = 😀 [an element of U]
- holds(Jon = Jack, I)

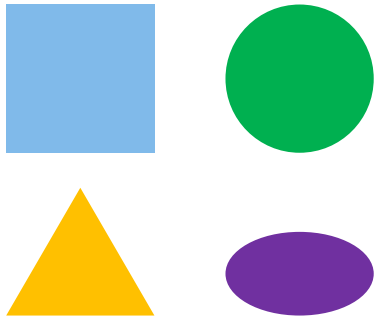
Semantics of Quantifiers

- Extend an interpretation I to bind variable x to element a ∈ U: $I_{x/a}$
  - holds(∀x.Φ), I) iff holds(Φ, $I_{x/a}$) for all a ∈ U
  - holds(∃x.Φ), I) iff holds(Φ, $I_{x/a}$) for some a ∈ U
- Quantifier applies to formula to right until an enclosing right parenthesis.

$$\left(\forall x.\, P(x) \ \vee Q(x)\right) \vee \ \exists x.\, R(x) \rightarrow Q(x)$$

FOL Example Domain

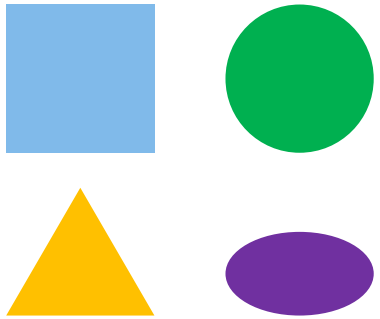
- U = { ■ , ● , ▲ , ⬭ }
- Constants
  - Fred
- Preds
  - $Above_2$, $Circle_1$, $Oval_1$, $Square_1$
- Function
  - hat
- I(Fred) = ▲



The real world

FOL Example Domain

- I(Fred) = ▲
- I(Above) = {<■, ▲>, <●, ⬭>}
- I(Circle) = {<●>}
- I(Oval) = {<●>, <⬭>}
- I(hat) = {<▲, ■>, <⬭, ●>,
  <■, ■>, <●, ●>}
- I(Square) = {<▲>}



The real world

FOL Example Domain

- I(Fred) = 
- I(Above) = {<  ,  >, <  ,  >}
- I(Circle) = {<  >}
- I(Oval) = {<  >, <  >}
- I(hat) = {<  ,  >, <  ,  >,

  <  ,  >, <  ,  >}
- I(Square) = {< >}

holds(Square(Fred), I)?

holds(Above(Fred, hat(Fred)), I)?

- I(hat(Fred)) = 
- holds(Above( , ), I)?

FOL Example Domain

- I(Fred) = 
- I(Above) = {<  ,  >, <  ,  >}
- I(Circle) = {<  >}
- I(Oval) = {<  >, <  >}
- I(hat) = {<  ,  >, <  ,  >,

  <  ,  >, <  ,  >}
- I(Square) = {< >}

holds(∃x.Oval(x), I)?

- holds(Oval(x), $I_{x/}$ )?

holds(∀x. ∃y. Above(x,y) ∨ Above(y, x), I)?

- holds(∃y. Above(x,y) ∨ Above(y, x), $I_{x/}$)?
  - holds(Above(x,y) ∨ Above(y, x), $I_{x/}$  $_{,y/}$ )?
- verify for all other values of x.

holds(∀x. ∀y. Above(x,y) ∨ Above(y, x), I)?

- holds(Above(x,y) ∨ Above(y, x), $I_{x/}$$_{,y/}$)?

Writing FOL

1. Cats are mammals. [Cat^1, $Mammal^1$]

- $\forall x.\ Cat(x) \rightarrow Mammal(x)$

2. Jane is a tall surveyor. [$Tall^1$, $Surveyor^1$, Jane]

- $Tall(Jane) \wedge Surveyor(Jane)$

3. A nephew is a sibling's son. [$Nephew^2$, $Sibling^2$, Son^2]

- $\forall xy.\ [Nephew(x,y) \leftrightarrow \exists z.\ [Sibling(y,z) \wedge Son(x,z)]]$

4. A maternal grandmother is a mother's mother. [functions: mgm, mother-of]

- $\forall xy.\ x=mgm(y) \rightarrow \exists z.\ x=mother\text{-}of(z) \wedge z=mother\text{-}of(y)$

Everybody loves somebody [loves2]

$\forall x.\ \exists y.\ Loves(x, y)$

$\exists y.\ \forall x.\ Loves(x, y)$

$\exists y.\ \exists x.\ Loves(x, y)$

$\forall xy.\ Loves(x, y)$

Writing More FOL

5. Nobody loves Jane.

- ∀x. ¬ Loves(x, Jane)
- ¬ ∃x. Loves(x, Jane)

6. Everybody has a father.

- ∀x. ∃y. Father(y,x)

7. Everybody has a father and a mother.

- ∀x. ∃yz. Father(y,x) ∧ Mother(z, x)

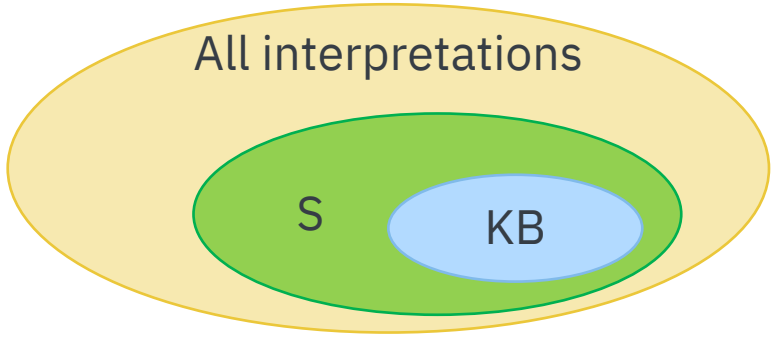
8. Whoever has a father, has a mother.

- ∀x. [[∃y. Father(y,x)] → [∃y. Mother(y,x)]]

Entailment in First-Order Logic

KB entails S

- **For every interpretation I, if KB holds in I, then S holds in I.**



- Computing entailment is impossible in general because there are infinitely many possible interpretations.
- Even computing holds is impossible for interpretations with infinite universes.

Intended Interpretations

KB: $(\forall x.\text{Circle}(x) \rightarrow \text{Oval}(x)) \wedge (\forall x.\text{Square}(x) \rightarrow \neg\, \text{Oval}(x))$

S: $\forall x.\text{Square}(x) \rightarrow \neg\, \text{Oval}(x)$

- We know holds(KB, I).
- We wonder whether holds(S, I).
- We could ask “Does KB entail S?”
- Or we could just try to check whether holds(S, I)

- I(Fred) = 
- I(Above) = {<  ,  >, <  ,  >}
- I(Circle) = {<  >}
- I(Oval) = {<  >, <  >}
- I(hat) = {<  , >, <  ,  >,
  <  ,  >, <  ,  >}
- I(Square) = {<  >}

An Infinite Interpretation

KB: $(\forall x.\text{Circle}(x) \rightarrow \text{Oval}(x)) \wedge (\forall x.\text{Square}(x) \rightarrow \neg\, \text{Oval}(x))$

S: $\forall x.\text{Square}(x) \rightarrow \neg\, \text{Oval}(x)$

- Does KB hold in I_1?
  - Yes, but can't verify by enumerating U.
- S also holds in I_1.
  - Yes, but can't verify by enumerating U.

- $U_1 = \{1, 2, 3, \ldots\}$
- I_1 (Circle) = {<4, 8, 12, 16, ...>}
- I_1 (Oval) = {2, 4, 6, 8, ...}
- I_1 (Square) = {1, 3, 5, 7, ...}

An argument for Entailment

KB: $(\forall x.\text{Circle}(x) \rightarrow \text{Oval}(x)) \wedge (\forall x.\text{Square}(x) \rightarrow \neg\, \text{Oval}(x))$

S_1: $\forall xy.\ (\text{Circle}(x) \wedge \text{Over}(y) \wedge \neg\text{Circle}(y)) \rightarrow \text{Above}(x,y)$

- I(Fred) = 
- I(Above) = {<  ,  >, <  ,  >}
- I(Circle) = {<  >}
- I(Oval) = {<  >, <  >}
- I(hat) = {<  ,  >, <  ,  >,
  <  ,  >, <  ,  >}
- I(Square) = {<  >}

- holds(KB, I)
- holds(S_1, I)

- U_1 = {1, 2, 3, ...}
- I_1 (Circle) = {<4, 8, 12, 16, ...>}
- I_1 (Oval) = {2, 4, 6, 8, ...}
- I_1 (Square) = {1, 3, 5, 7, ...}
- I_1 (Above) = >

- holds(KB, I_1)
- fails(S_1, I_1)

KB doesn't entail S_1!

Proof and Entailment

- Entailment captures general notion of “follows from.”
- We can’t evaluate it directly by enumerating interpretations.
- So, we will do proofs.
- In FOL, if S is entailed by KB, then there is a finite proof of S from KB.

Axiomatization

- What if we have a particular interpretation, I, in mind, and want to test whether holds(S, I)?
- Write down a set of sentences, called axioms, that will serve as our KB.
- We would like KB to hold in I, and as few other interpretations as possible.
- No matter what,
  - If holds(KB, I) and KB entails S, then holds (S, I).
- If your axioms are week, it might be that
  - holds(KB, I) and holds(S, I), but
  - KB doesn’t entail S.

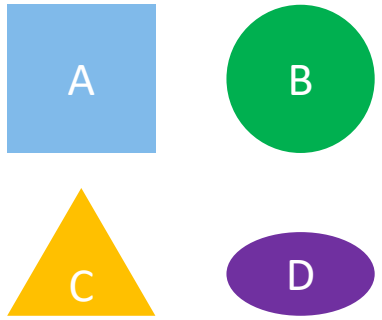
Axiomatization Example

KB_2

- Above(A, C)
- Above(B, D)
- ∀xy. Above(x,y) → hat(y) = x
- ∀x.(¬∃y. Above(y,x)) → hat(x) = x

S

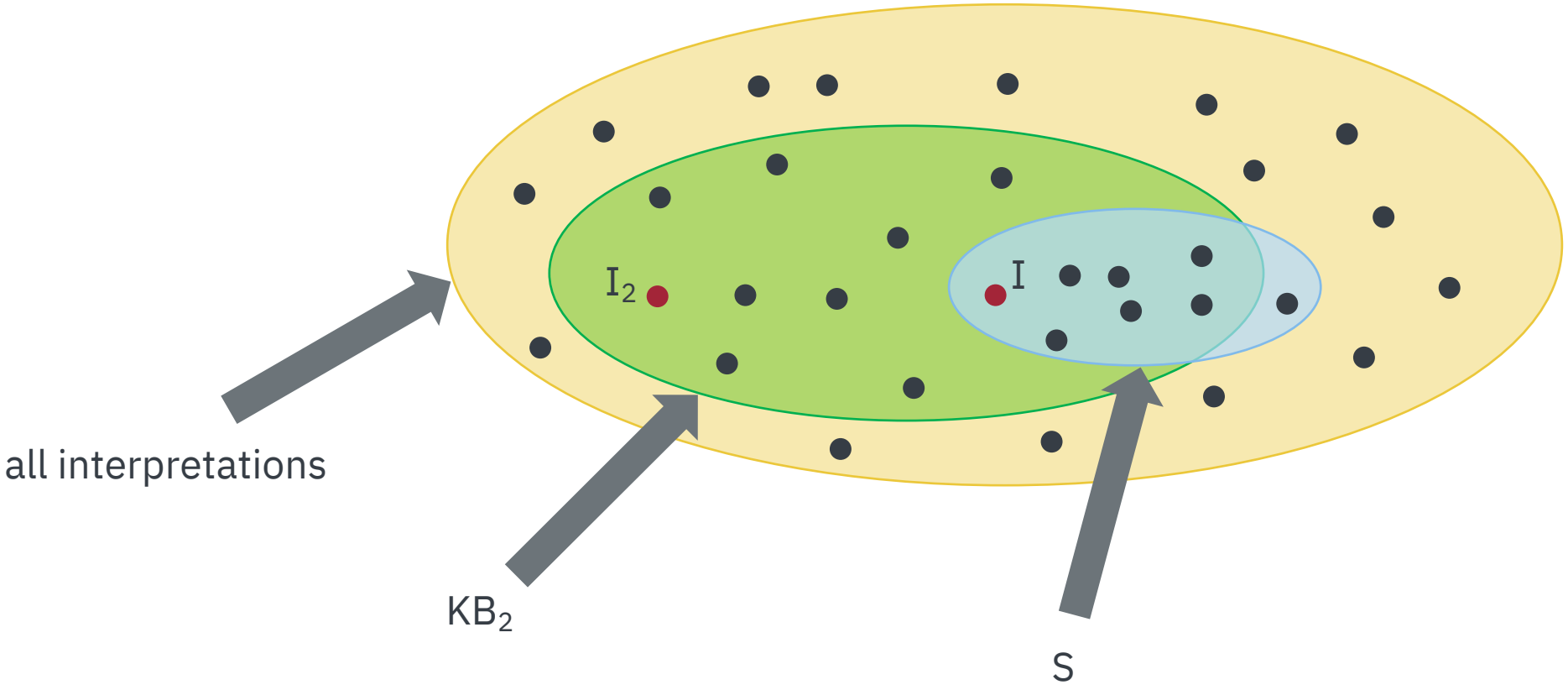
- hat(A) = A



- I_2(A) = ■
- I_2(B) = ●
- I_2(C) = ▲
- I_2(D) = ⬭
- I_2(Above) = {<■,▲>, <●, ⬭>, <▲, ■>, <⬭, ●>}
- I_2(hat) = {<▲,■>, <⬭,●>, <●, ⬭>, <■,▲>}

- holds(KB_2, I_2)
- fails(S, I_2)
- KB_2 doesn't entail S.

KB_2 is a Weakling!



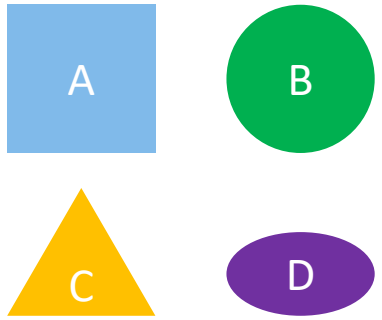
Axiomatization Example: Another Try

KB_3

- Above(A, C)
- Above(B, D)
- ∀xy. Above(x,y) → hat(y) = x
- ∀x.(¬∃y. Above(y,x)) → hat(x) = x
- **∀xy. Above(x,y) → ¬Above(y,x)**

S

- hat(A) = A



- I_3(A) = ■
- I_3(B) = ●
- I_3(C) = ▲
- I_3(D) = ⬭
- I_3(Above) = {<■,▲>, <●, ⬭>, <●, ■>}
- I_3(hat) = {<▲,■>, <⬭,●>, <●,●>, <■,●>}

- fails (KB_3, I_2)
- holds(KB_3, I_3)
- fails(S, I_3)
- KB_3 doesn't entail S.

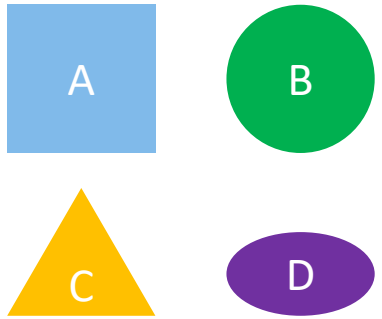
Axiomatization Example: Another Try

KB_4

- Above(A, C)
- Above(B, D)
- **¬∃x. Above(x, A)**
- **¬∃x. Above(x, B)**
- ∀xy. Above(x,y) → hat(y) = x
- ∀x.(¬∃y. Above(y,x)) → hat(x) = x

S

- hat(A) = A



- I_3(A) = ■
- I_3(B) = ●
- I_3(C) = ▲
- I_3(D) = ⬭
- I_3(Above) = {<■,▲>, <●, ⬭>, <●, ■>}
- I_3(hat) = {<▲,■>, <⬭,●>, <●,●>, <■,●>}

- fails (KB_4, I_3)
- KB_4 entails S.

We will prove KB_4 entails S.

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Questions?

Acknowledgement

Fahiem Bacchus, University of Toronto
Dan Klein, UC Berkeley
Kate Larson, University of Waterloo