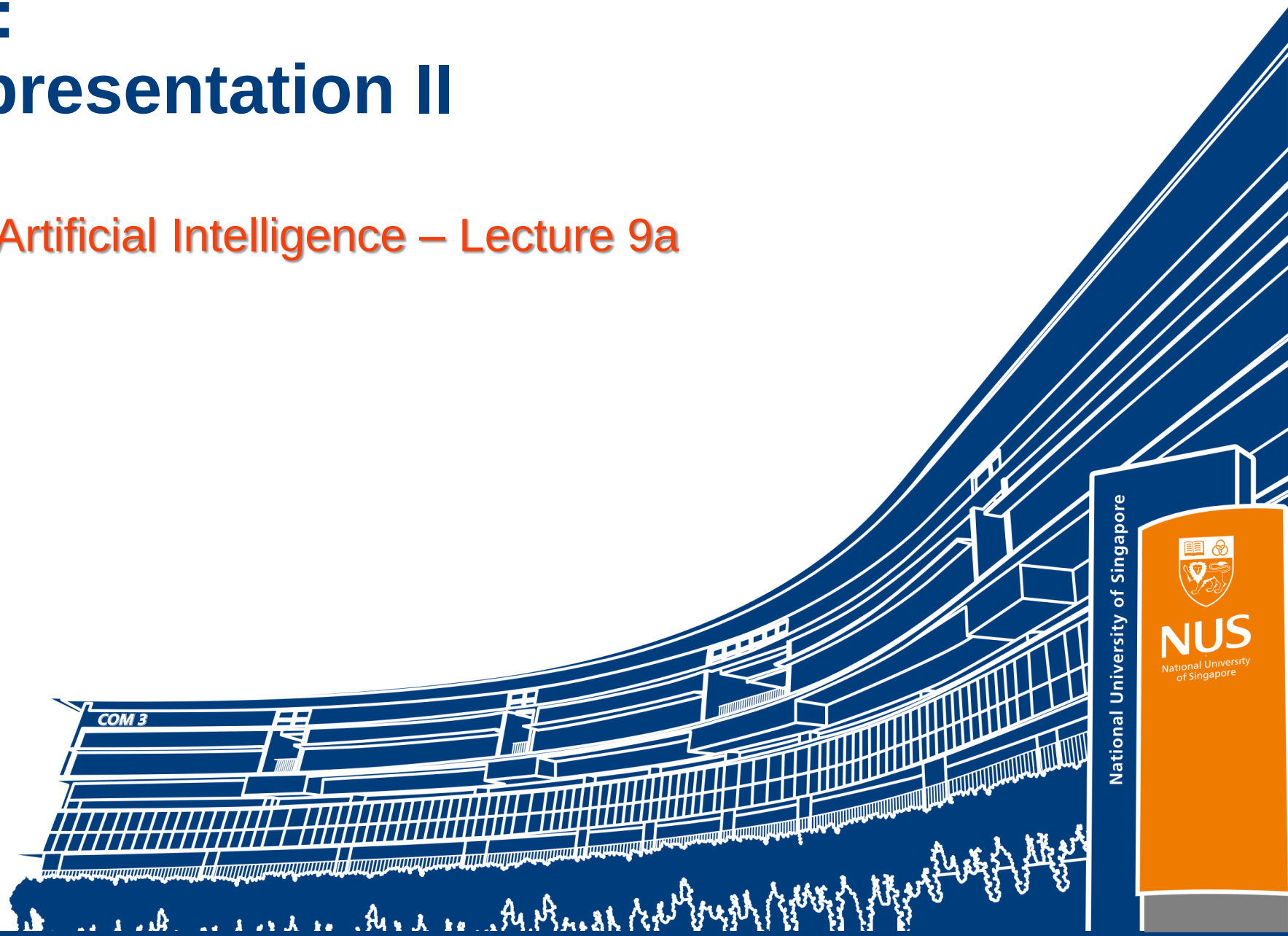
Logical Agents: Knowledge Representation II

CS3243: Introduction to Artificial Intelligence – Lecture 9a



NUS | Computing
National University of Singapore

1 Administrative Matters

Upcoming...

- Deadlines
  - Tutorial Assignment 7 (released last week)
    - Post-tutorial submission: Due this Friday!
  - Tutorial Assignment 8 (released today)
    - Pre-tutorial Submission: Due this Sunday!
    - Post-tutorial Submission: Due next Friday!
  - Project 3 (released 14 October)
    - Due in Week 12

    12 November 2024

Final Content Lecture next week!

- No Lecture in Week 12
- Course Review Lecture in Week 13
  - Includes review of Final Exam Paper Structure

Project Consultations:
Thursdays (1900 hrs) via Zoom

Contents

- Recap on Logical Agents
- Theorem-Proving Methods
- Resolution
- Uncertainty & Recap on Probability

2 Recap on Logical Agents

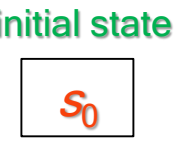
Logical Agents

- Agent contains
  - Knowledge Base (KB)
    - Specified in some language (e.g., propositional logic)
  - Inference Engine (IE)
    - Based on sentences that will guide action choice, $\alpha_1, \alpha_2, \ldots, \alpha_k$
    - Uses an algorithm that infers α_i such the KB $\models \alpha_i$
  - General algorithm
    - Pre-populate KB with domain knowledge
    - Each time step t :
      - Update KB with percepts
      - Use IE to make inferences
      - Update KB with inferences
      - Select and execute action based on inferences

Remember that there are two forms of "search" taking place
- **Solving the problem environment**
- **Solving the inference problems (α)**

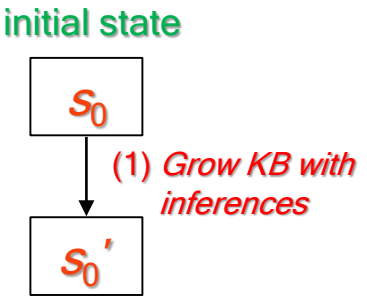
Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows



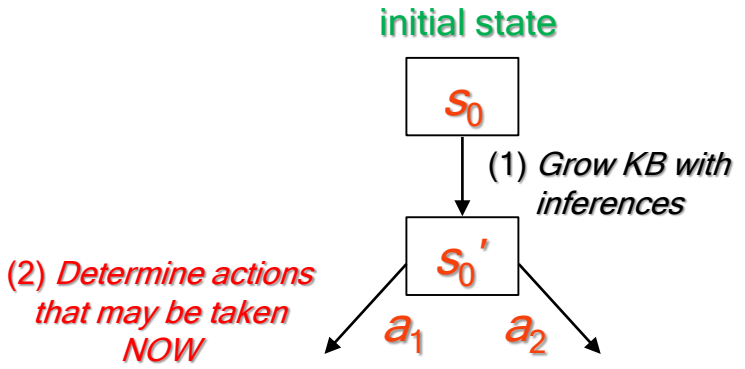
Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows
  - Make inferences about environment
    - Assume query (α_i) to action (a_i) mappings
      - e.g., reflex agent with conditions based on KB
    - Developer designing the agent must formulate these mappings



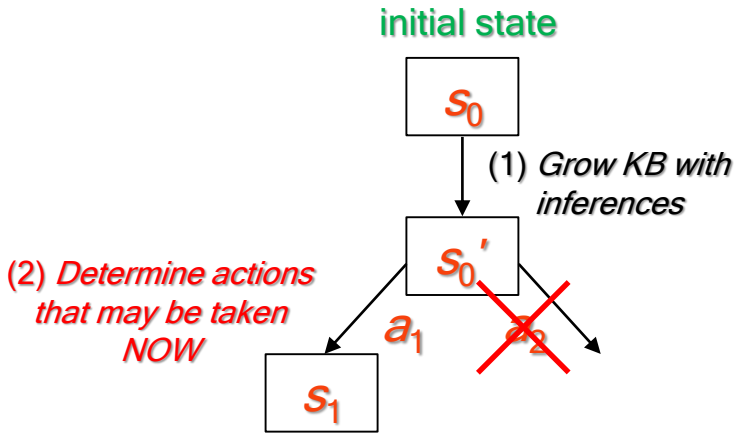
Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows
  - Make inferences about environment
    - Assume query (α_i) to action (a_i) mappings
      - e.g., reflex agent with conditions based on KB
    - Developer designing the agent must formulate these mappings
  - Suppose $\alpha_i \Rightarrow a_i$
    - Agent uses the KB & IE to determine if a query (α_i) may be inferred, and if so, if an action (a_i) may be taken



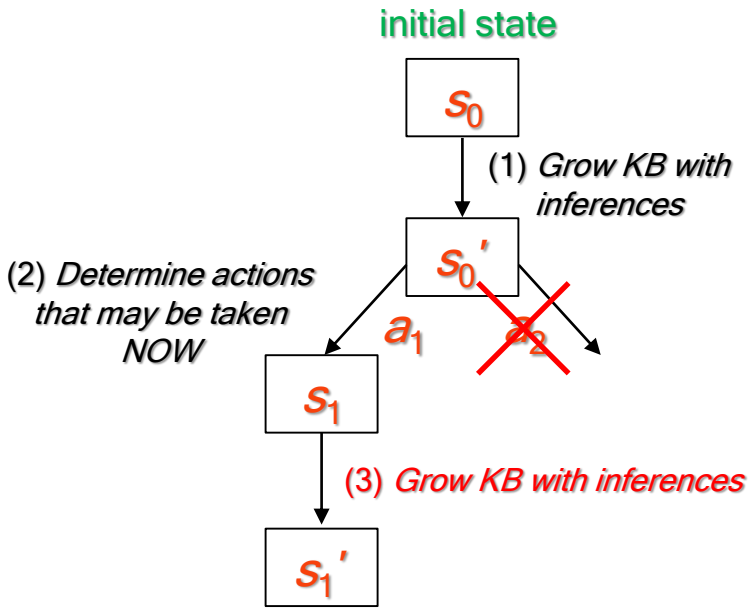
Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows
  - Make inferences about environment
    - Assume query (α_i) to action (a_i) mappings
      - e.g., reflex agent with conditions based on KB
    - Developer designing the agent must formulate these mappings
  - Suppose $\alpha_i \Rightarrow a_i$
    - Agent uses the KB & IE to determine if a query (α_i) may be inferred, and if so, if an action (a_i) may be taken
    - Chosen action (a_i) is EXECUTED in environment (as we are no longer planning)



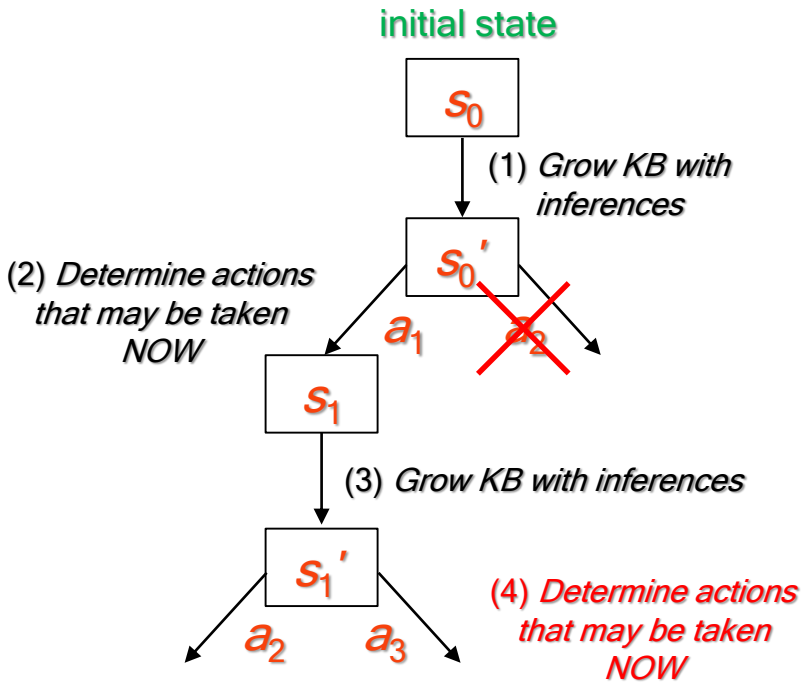
Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows
  - Make inferences about environment
    - Assume query (α_i) to action (a_i) mappings
      - e.g., reflex agent with conditions based on KB
    - Developer designing the agent must formulate these mappings
  - Suppose $\alpha_i \Rightarrow a_i$
    - Agent uses the KB & IE to determine if a query (α_i) may be inferred, and if so, if an action (a_i) may be taken
      - Chosen action (a_i) is EXECUTED in environment (as we are no longer planning)
  - Loop...



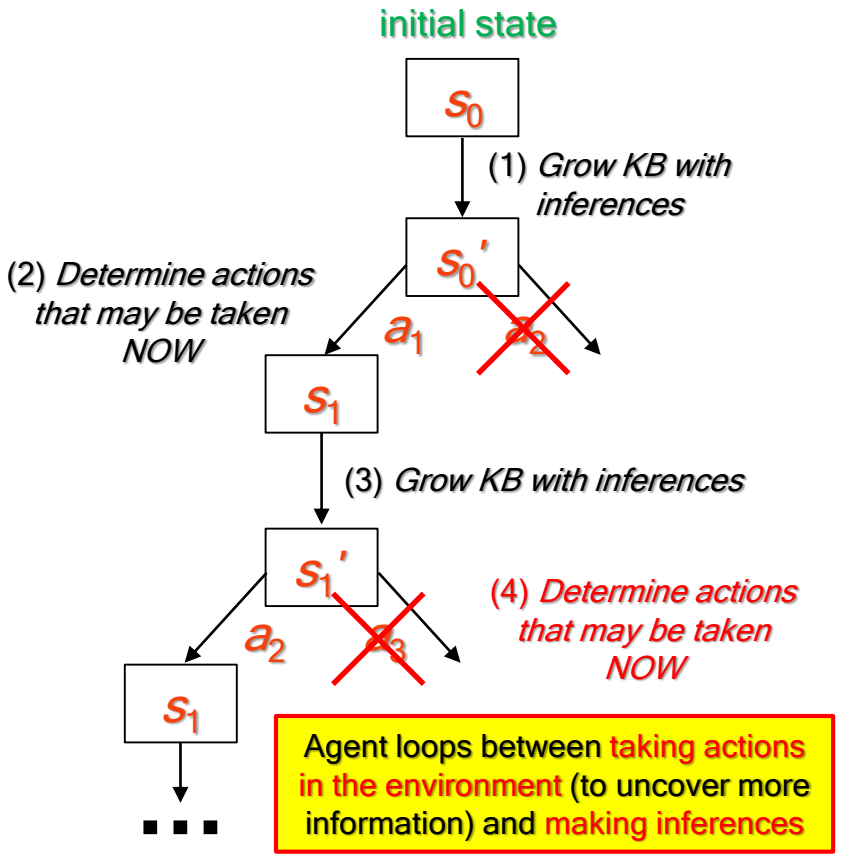
Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows
  - Make inferences about environment
    - Assume query (α_i) to action (a_i) mappings
      - e.g., reflex agent with conditions based on KB
    - Developer designing the agent must formulate these mappings
  - Suppose $\alpha_i \Rightarrow a_i$
    - Agent uses the KB & IE to determine if a query (α_i) may be inferred, and if so, if an action (a_i) may be taken
      - Chosen action (a_i) is EXECUTED in environment (as we are no longer planning)
  - Loop...



Logical Agents

- Logical Agent cannot plan an entire path to goal
  - Environment is only Partially Observable
- Assume operation as follows
  - Make inferences about environment
    - Assume query (α_i) to action (a_i) mappings
      - e.g., reflex agent with conditions based on KB
    - Developer designing the agent must formulate these mappings
  - Suppose $\alpha_i \Rightarrow a_i$
    - Agent uses the KB & IE to determine if a query (α_i) may be inferred, and if so, if an action (a_i) may be taken
      - Chosen action (a_i) is EXECUTED in environment (as we are no longer planning)
  - Loop…



Making Inferences

- Entailment ($\models$)
  - $KB \models \alpha$ means that $M(KB) \subseteq M(\alpha)$
    - This says that all value assignments that satisfy the KB will also satisfy α
    - i.e., whenever KB is true, α is true
- Inference algorithm ($\mathcal{A}$)
  - Sound: $(KB \vdash_{\mathcal{A}} \alpha) \Rightarrow (KB \models \alpha)$
    - $\mathcal{A}$ only infers α that are valid
  - Complete: $(KB \models \alpha) \Rightarrow (KB \vdash_{\mathcal{A}} \alpha)$
    - $\mathcal{A}$ is able to infer all valid α
- Inference algorithm example: Truth Table Enumeration
  - Constructs entire truth table for KB and α
  - Check (via DFS) that $M(KB) \subseteq M(\alpha)$
  - i.e., every model of KB is a model of α

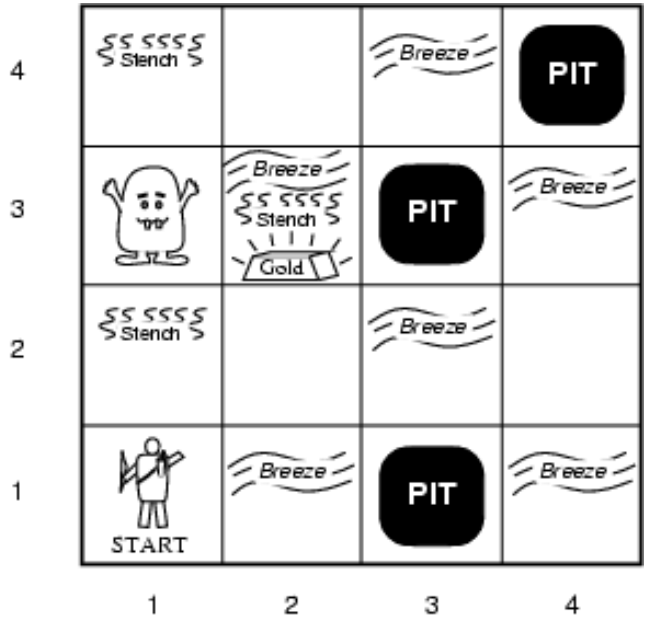
Truth Table Enumeration:
- Sound and Complete
- Time complexity $O(2^n)$
- Space complexity $O(n)$

Lecture Exercise: Wumpus World

- What is the relevant agent function for Wumpus World?

```
// assume global KB
s, gold, safe, togo = (1,1), False, {(1,1),}, {}
while not gold:
```

?




```
while not at_exit(s):
      goto_exit(safe)
      // i.e., solve path search to
      // exit only using safe cells
exit_dungeon()
```

3 Theorem Proving Methods

Proof Methods

- Model checking (special case of CSPs where domains are T/F)
  - Truth Table Enumeration (brute-force proof - time complexity exponential in n)

- Applying inference rules (i.e., theorem proving)
  - Generate new sentences from old
  - Proof = sequential application of inference rules
    - Inference rules help deduce valid actions
    - Proof facilitates efficiency – ignores irrelevant propositions

  - Resolution (inference via proof by contradiction)

Validity & Satisfiability

- A sentence α is valid if it is true for ALL possible truth value assignments
  - e.g., True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
  - i.e., tautologies

Validity & Satisfiability

- A sentence α is valid if it is true for ALL possible truth value assignments
  - e.g., True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
  - i.e., tautologies
- Validity is connected to entailment via the Deduction Theorem:
  - $(KB \models \alpha) \Leftrightarrow (KB \Rightarrow \alpha)$ is valid

Validity & Satisfiability

- A sentence α is valid if it is true for ALL possible truth value assignments
  - e.g., True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
  - i.e., tautologies
- Validity is connected to entailment via the Deduction Theorem:
  - $(KB \models \alpha) \Leftrightarrow (KB \Rightarrow \alpha)$ is valid
- A sentence is satisfiable if it is true for SOME truth value assignment → i.e., a model exists for that sentence
  - e.g., $A \vee B$, C
- A sentence if unsatisfiable if it is true for NO truth value assignments → i.e., no model exists for that sentence
  - e.g., $A \wedge \neg A$
  - i.e., contradictions

Validity & Satisfiability

- A sentence α is valid if it is true for ALL possible truth value assignments
  - e.g., True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
  - i.e., tautologies
- Validity is connected to entailment via the Deduction Theorem:
  - $(KB \models \alpha) \Leftrightarrow (KB \Rightarrow \alpha)$ is valid
- A sentence is satisfiable if it is true for SOME truth value assignment → i.e., a model exists for that sentence
  - e.g., $A \vee B$, C
- A sentence if unsatisfiable if it is true for NO truth value assignments → i.e., no model exists for that sentence
  - e.g., $A \wedge \neg A$
  - i.e., contradictions
- Satisfiability is connected to entailment via the following:
  - $(KB \models \alpha) \Leftrightarrow (KB \wedge \neg\alpha)$ is unsatisfiable
  - i.e., definition of Proof by Contradiction

- $A \Rightarrow B \equiv \neg A \vee B$
- $\neg(\neg A \vee B) \equiv A \wedge \neg B$
- Showing that $(A \wedge \neg B)$ is unsatisfiable shows that the negation, $\neg A \vee B$, is valid!

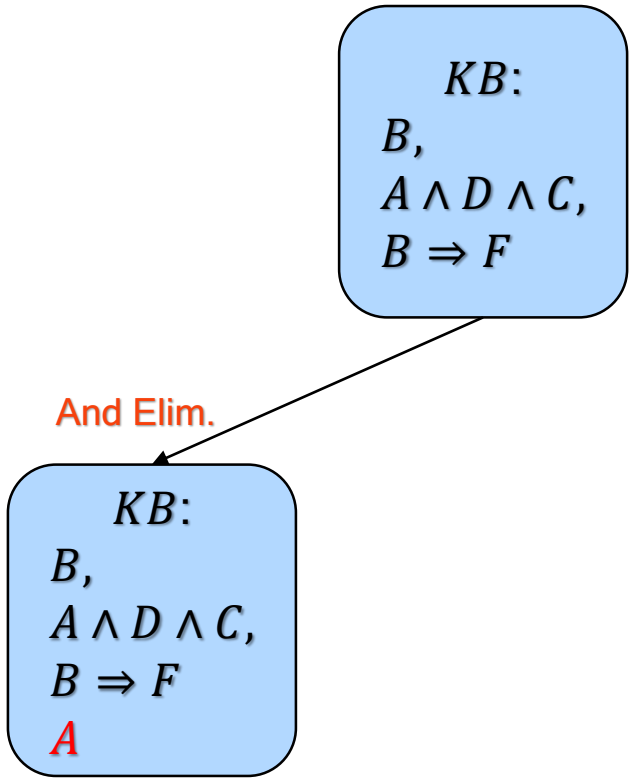
Inference Algorithms: Application of Inference Rules

- Search for more knowledge (growing the KB)
  - Equivalent to a search problem
    - States: Versions of the KB (e.g., initial state is initial KB)
    - Actions: Application of inference rules
    - Transition: Update KB with an inferred sentence (may not be target query α)
    - Goal: KB contains sentence to (dis)prove α
- Examples of inference rules

KB:
B,
$A \wedge D \wedge C$,
$B \Rightarrow F$

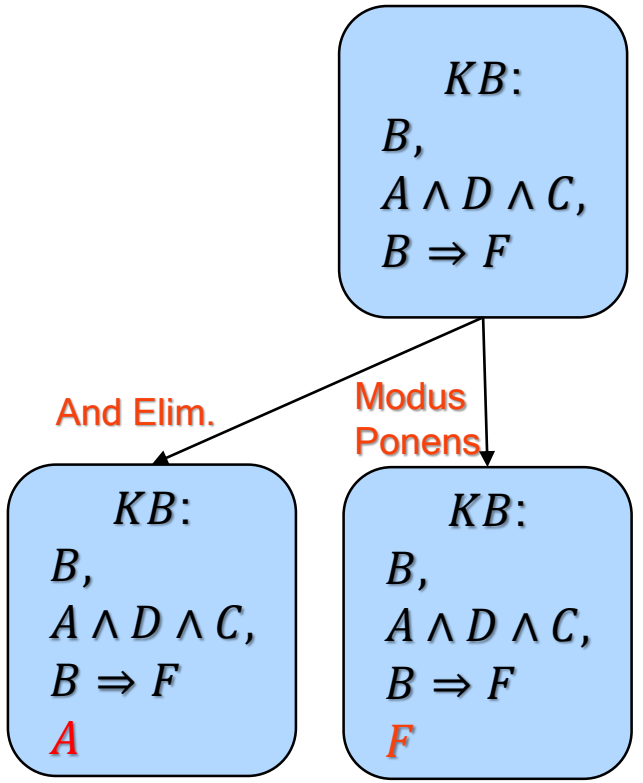
Inference Algorithms: Application of Inference Rules

- Search for more knowledge (growing the KB)
  - Equivalent to a search problem
    - States: Versions of the KB (e.g., initial state is initial KB)
    - Actions: Application of inference rules
    - Transition: Update KB with an inferred sentence (may not be target query α)
    - Goal: KB contains sentence to (dis)prove α
- Examples of inference rules
  - And-Elimination (AE): e.g., $a \wedge b \vDash a$; $a \wedge b \vDash b$



Inference Algorithms: Application of Inference Rules

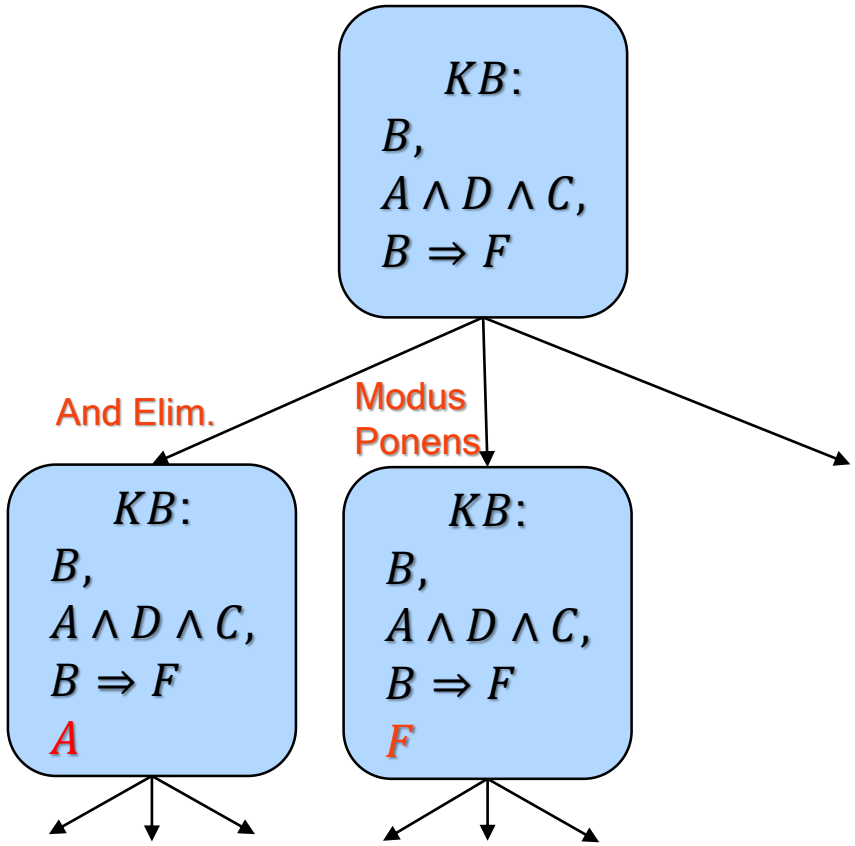
- Search for more knowledge (growing the KB)
  - Equivalent to a search problem
    - States: Versions of the KB (e.g., initial state is initial KB)
    - Actions: Application of inference rules
    - Transition: Update KB with an inferred sentence (may not be target query α)
    - Goal: KB contains sentence to (dis)prove α
- Examples of inference rules
  - And-Elimination (AE): e.g., $a \wedge b \vDash a; a \wedge b \vDash b$
  - Modus Ponens (MP): e.g., $a \wedge (a \Rightarrow b) \vDash b$



Inference Algorithms: Application of Inference Rules

- Search for more knowledge (growing the KB)
  - Equivalent to a search problem
    - States: Versions of the KB (e.g., initial state is initial KB)
    - Actions: Application of inference rules
    - Transition: Update KB with an inferred sentence (may not be target query α)
    - Goal: KB contains sentence to (dis)prove α
- Examples of inference rules
  - And-Elimination (AE): e.g., $a \wedge b \vDash a; a \wedge b \vDash b$
  - Modus Ponens (MP): e.g., $a \wedge (a \Rightarrow b) \vDash b$
  - Logical Equivalences: e.g., $(a \vee b) \vDash \neg(\neg a \wedge \neg b)$

How does this relate to Truth Table Enumeration?



4 Resolution

Resolution for Conjunctive Normal Form (CNF)

- CNF = conjunction of disjunctive sentences
  - e.g., $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$
- Resolution
  - Method of simplifying KB to prove entailment of query α
  - Specifically:
    - Given KB: $R_1 \wedge R_2 \wedge \ldots \wedge R_n$
    - If a literal, x, appears in R_i and its negation, $\neg x$, appears in R_j, where $R_i, R_j \in$ KB, it can be removed from both!

resolvent

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

Resolution for Conjunctive Normal Form (CNF)

- CNF = conjunction of disjunctive sentences
  - e.g., $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$
- Resolution
  - Method of simplifying KB to prove entailment of query α
  - Specifically:
    - Given KB: $R_1 \wedge R_2 \wedge \ldots \wedge R_n$
    - If a literal, x, appears in R_i and its negation, $\neg x$, appears in R_j, where $R_i, R_j \in$ KB, it can be removed from both!

resolvent

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

KB: $(P \vee x) \wedge (Q \vee \neg x)$
α: $(P \vee Q)$ must hold?

x=True, Q must be True for KB to hold
x=False, P must be True for KB to hold

Assuming KB=True, $(P \vee Q)$ must hold!

Verify with truth table as an exercise
(note that we want M(KB) ⊆ M(α);
not KB ≡ α)

- Resolution under propositional logic is both sound and complete

Some Rules for Conversion to CNF

- Convert:
  1. $\alpha \Leftrightarrow \beta$ to $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  2. $\alpha \Rightarrow \beta$ to $\neg\alpha \vee \beta$

- Expand ¬ using De Morgan's and double negation; i.e., convert:
  3. $\neg(\alpha \vee \beta)$ to $\neg\alpha \wedge \neg\beta$
  4. $\neg(\alpha \wedge \beta)$ to $\neg\alpha \vee \neg\beta$
  5. $\neg(\neg\alpha)$ to α

- Convert:
  6. $(\alpha \vee (\beta \wedge \gamma))$ to $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Some Rules for Conversion to CNF

- Convert:
  1. $\alpha \Leftrightarrow \beta$ to $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  2. $\alpha \Rightarrow \beta$ to $\neg\alpha \vee \beta$
- Expand ¬ using De Morgan's and double negation; i.e., convert:
  3. $\neg(\alpha \vee \beta)$ to $\neg\alpha \wedge \neg\beta$
  4. $\neg(\alpha \wedge \beta)$ to $\neg\alpha \vee \neg\beta$
  5. $\neg(\neg\alpha)$ to α
- Convert:
  6. $(\alpha \vee (\beta \wedge \gamma))$ to $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Each of these conversions produces two rules in the KB (the others only produce one)

Resolution Algorithm

- Utilises proof by contradiction – tries to show that KB ∧ ¬α is unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Resolution Algorithm

- Utilises proof by contradiction – tries to show that KB ∧ ¬α is unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Suppose we have a KB as follows:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

Resolution Algorithm

- Utilises proof by contradiction – tries to show that KB ∧ ¬α is unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Suppose we have a KB as follows:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

And the algorithm slowly removes literals:

$$\frac{(\cancel{x_1 \vee \cdots \vee x_m} \vee x) \wedge (\cancel{y_1 \vee \cdots \vee y_k} \vee \neg x)}{(\cancel{x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k})}$$

Resolution Algorithm

- Utilises proof by contradiction – tries to show that KB ∧ ¬α is unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Suppose we have a KB as follows:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

And the algorithm slowly removes literals:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

What does an empty clause imply??

Resolution Algorithm

- Utilises proof by contradiction – tries to show that KB ∧ ¬α is unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

What does an empty clause imply??

Suppose we have a KB as follows:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

And the algorithm slowly removes literals:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

Eventually, if there is an empty clause, it indicates that the disjunction of no literals holds. A disjunction is True only when at least one literal is true.

So, whole KB is False here since the KB is a conjunction of all rules including the empty clause - i.e., the query ¬α is unsatisfiable.

We may infer α (via proof by contradiction).

Resolution Algorithm

- Utilises proof by contradiction – tries to show that KB ∧ ¬α is unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

If cannot be resolved further and not empty clause then cannot infer α

What does an empty clause imply??

Suppose we have a KB as follows:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

And the algorithm slowly removes literals:

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

Eventually, if there is an empty clause, it indicates that the disjunction of no literals holds. A disjunction is True only when at least one literal is true.

So, whole KB is False here since the KB is a conjunction of all rules including the empty clause - i.e., the query ¬α is unsatisfiable.

We may infer α (via proof by contradiction).

Resolution Algorithm

- Summary
  - Make a clause list – i.e., copy of KB specified in CNF including negation of query, ¬α
    - Use conversion rules to convert KB to CNF
  - Repeatedly resolve two clauses from clause list
    - Add resolvent to clause list
  - Keep doing this till empty clause found or no more resolutions possible
    - If empty clause then can infer α
    - If no more resolutions and not empty clause then cannot infer α

Resolution Algorithm

- Summary
  - Make a clause list – i.e., copy of KB specified in CNF including negation of query, ¬α
    - Use conversion rules to convert KB to CNF
  - Repeatedly resolve two clauses from clause list
    - Add resolvent to clause list
  - Keep doing this till empty clause found or no more resolutions possible
    - If empty clause then can infer α
    - If no more resolutions and not empty clause then cannot infer α

Why is Resolution under Propositional Logic Sound and Complete?

Soundness:
- Each resolvent is implied by generating clauses
- If ∅ is found, then (KB ∧ ¬α) is unsatisfiable

Completeness:
- Based on the idea of resolution closure - set of all clauses derivable
- Not covered in CS3243
- Refer to AIMA 4th Edition pp. 228-229

Questions about the Lecture?

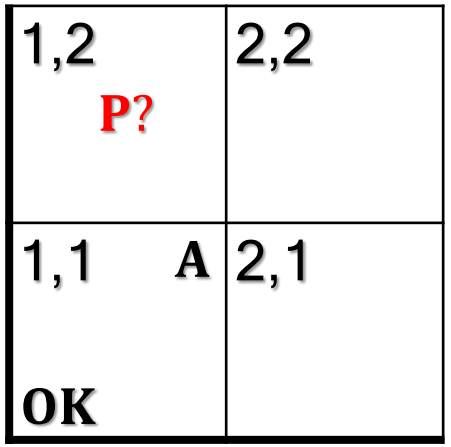
- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

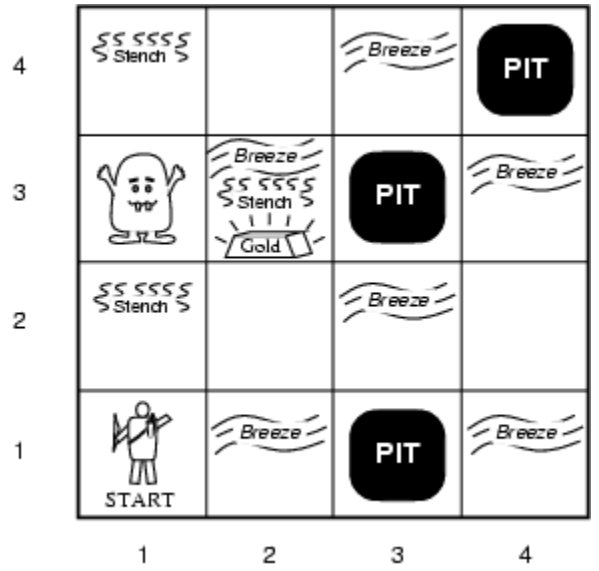
Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/64539863216

5 Resolution Example

Resolution Example: Back to Wumpus World

- Assume that agent is at (1,1) in Wumpus World
  - And we wish to make inferences about a pit at (1,2)
  - KB: $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge (\neg B_{1,1})$
    - i.e., we know:
      - R_a: rule for breezes
      - R_b: no breeze at (1,1)
  - $\alpha = \neg P_{1,2}$
    - i.e., want to know if we can safely move to (1,2)
  - Can we infer α?
    - Use the resolution algorithm to determine if (KB ⊨ α)
    - i.e., use (KB ⊨ α) ⇔ (KB ∧ ¬α) is unsatisfiable

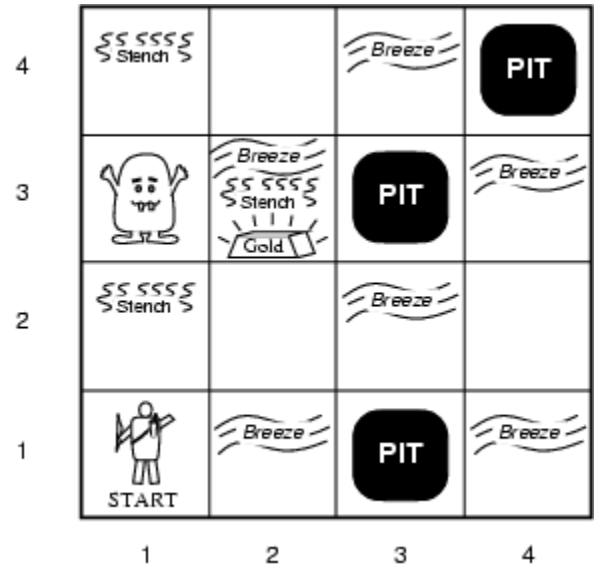




Resolution Example: Back to Wumpus World

- Given:
  - $KB = \neg B_{1,1} \wedge B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $\alpha = \neg P_{1,2}$

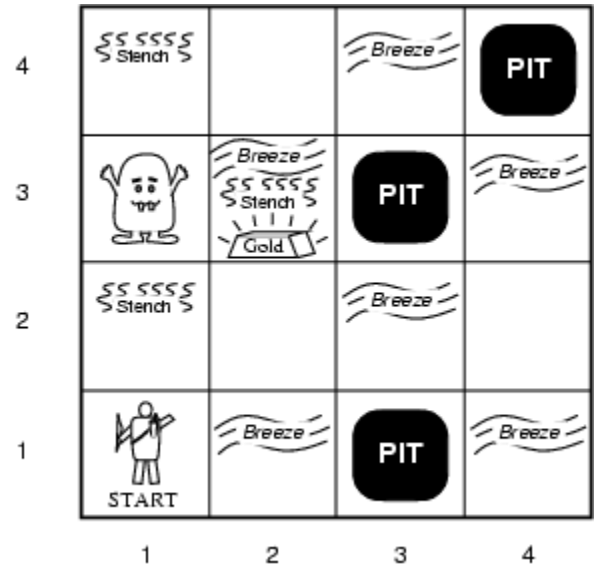
1,2 **P?**	2,2
1,1 **A** **OK**	2,1



Resolution Example: Back to Wumpus World

- Given:
  - $KB = \neg B_{1,1} \wedge B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $\alpha = \neg P_{1,2}$
- Step 1 – form clause list (over $KB \wedge \neg\alpha$):
  - $(\neg B_{1,1}) \wedge (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge (P_{1,2})$

1,2 **P?**	2,2
1,1 **A** **OK**	2,1

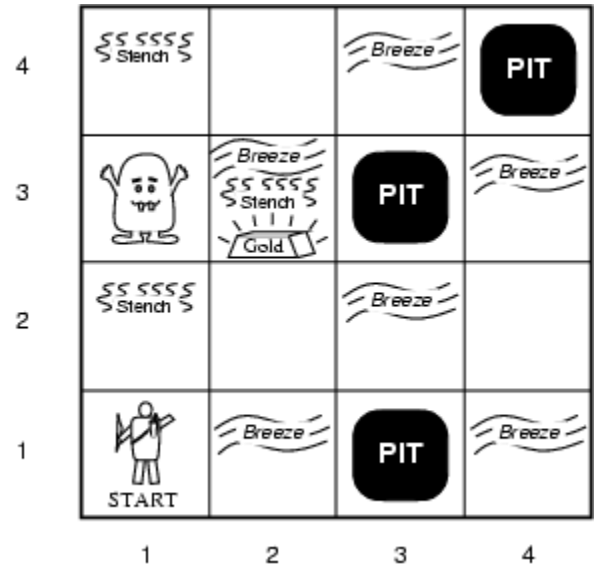


Resolution Example: Back to Wumpus World

- Given:
  - $KB = \neg B_{1,1} \wedge B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $\alpha = \neg P_{1,2}$
- Step 1 – form clause list (over $KB \wedge \neg\alpha$):
  - $(\neg B_{1,1}) \wedge (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge (P_{1,2})$
- Step 2 – Convert clause list to CNF:
  - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
    - $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$
    - $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$

$(\neg B_{1,1})$, $(P_{1,2})$ already in CNF (literals)

1,2 **P?**	2,2
1,1 **A** **OK**	2,1



Resolution Example: Back to Wumpus World

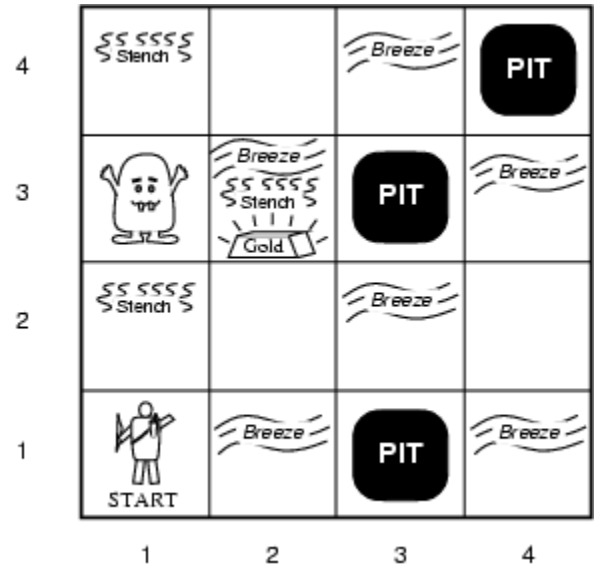
- Given:
  - $KB = \neg B_{1,1} \wedge B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $\alpha = \neg P_{1,2}$
- Step 1 – form clause list (over $KB \wedge \neg\alpha$):
  - $(\neg B_{1,1}) \wedge (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge (P_{1,2})$
- Step 2 – Convert clause list to CNF:
  - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
    - $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$
    - $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$

$(\neg B_{1,1})$, $(P_{1,2})$ already in CNF (literals)

- Step 2a:
  - $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$
    - $\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})$
    - $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$

Now in CNF

1,2 **P?**	2,2
1,1 **A** **OK**	2,1



Resolution Example: Back to Wumpus World

- Given:
  - $KB = \neg B_{1,1} \wedge B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $\alpha = \neg P_{1,2}$
- Step 1 – form clause list (over $KB \wedge \neg\alpha$):
  - $(\neg B_{1,1}) \wedge (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge (P_{1,2})$
- Step 2 – Convert clause list to CNF:
  - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
    - $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$
    - $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$

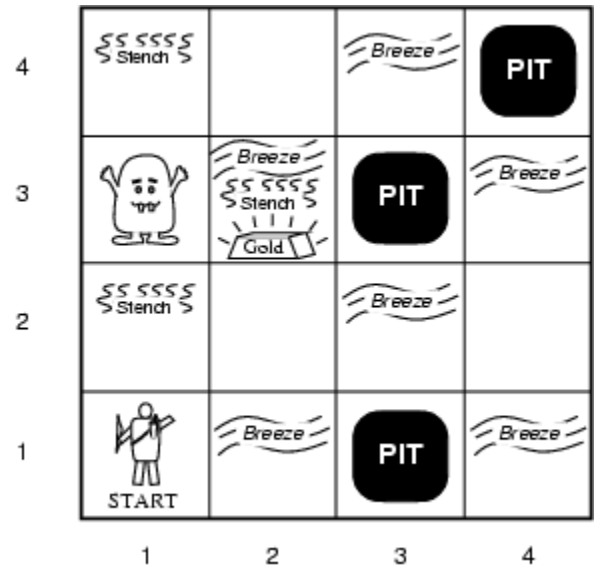
$(\neg B_{1,1})$, $(P_{1,2})$ already in CNF (literals)

- Step 2a:
  - $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$
    - $\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})$
    - $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$

Now in CNF

-

1,2 **P?**	2,2
1,1 **A** **OK**	2,1



- Step 2b:
  - $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$
    - $\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}$
    - $(\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}$
    - $(\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Now in CNF - as 2 rules

Resolution Example: Back to Wumpus World

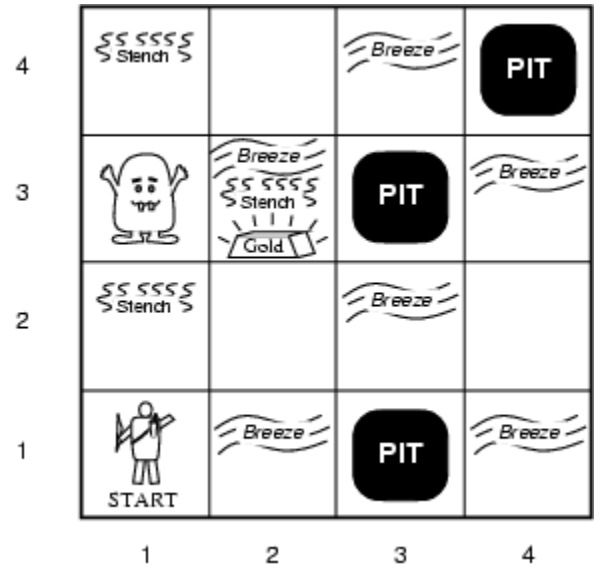
- Clause list (in CNF)
  - R_1: $\neg B_{1,1}$
  - R_2: $P_{1,2}$
  - R_3: $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
  - R_4: $\neg P_{1,2} \vee B_{1,1}$
  - R_5: $\neg P_{2,1} \vee B_{1,1}$
- Step 3 – Pick two rules and resolve via

$$\frac{(x_1 \vee \cdots \vee x_m \vee x) \wedge (y_1 \vee \cdots \vee y_k \vee \neg x)}{(x_1 \vee \cdots \vee x_m \vee y_1 \vee \cdots \vee y_k)}$$

- Step 3a – Reduce R_2 and R_4
  - R_6: $B_{1,1}$
- Step 3b – Reduce R_1 and R_6
  - ∅

Proof by contradiction that KB ⊨ α
i.e., α holds when KB holds; we can infer α = $\neg P_{1,2}$

1,2 **P?**	2,2
1,1 **A** **OK**	2,1



6 Where Does α Come From?

Regarding the Query α

- Inference algorithms show that we can infer α
- Where do we get α?

Regarding the Query α

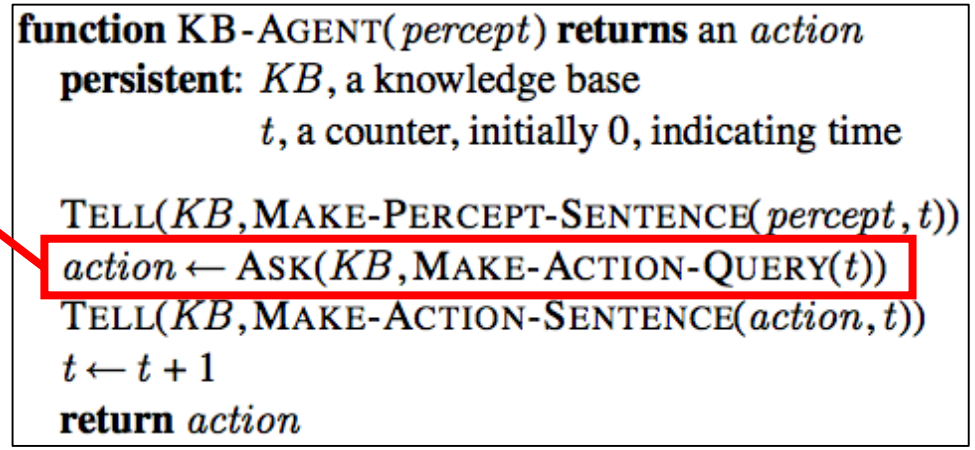
- Inference algorithms show that we can infer α
- Where do we get α?
  - Recall that Logical Agent program

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Regarding the Query α

- Inference algorithms show that we can infer α
- Where do we get α?
  - Recall that Logical Agent program

reasoning on what should be done



function KB-AGENT(*percept*) **returns** an *action*
  persistent: *KB*, a knowledge base
    t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  t ← *t* + 1
  return *action*

Regarding the Query α

- Inference algorithms show that we can infer α
- Where do we get α?
  - Recall that Logical Agent program

reasoning on what should be done

When modelling the problem, constraint (i.e., query) to action mappings must be determined by the developer

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

construct a sentence relating to an action to take

Regarding the Query α

- Inference algorithms show that we can infer α
- Where do we get α?
  - Recall that Logical Agent program

reasoning on what should be done

When modelling the problem, constraint (i.e., query) to action mappings must be determined by the developer

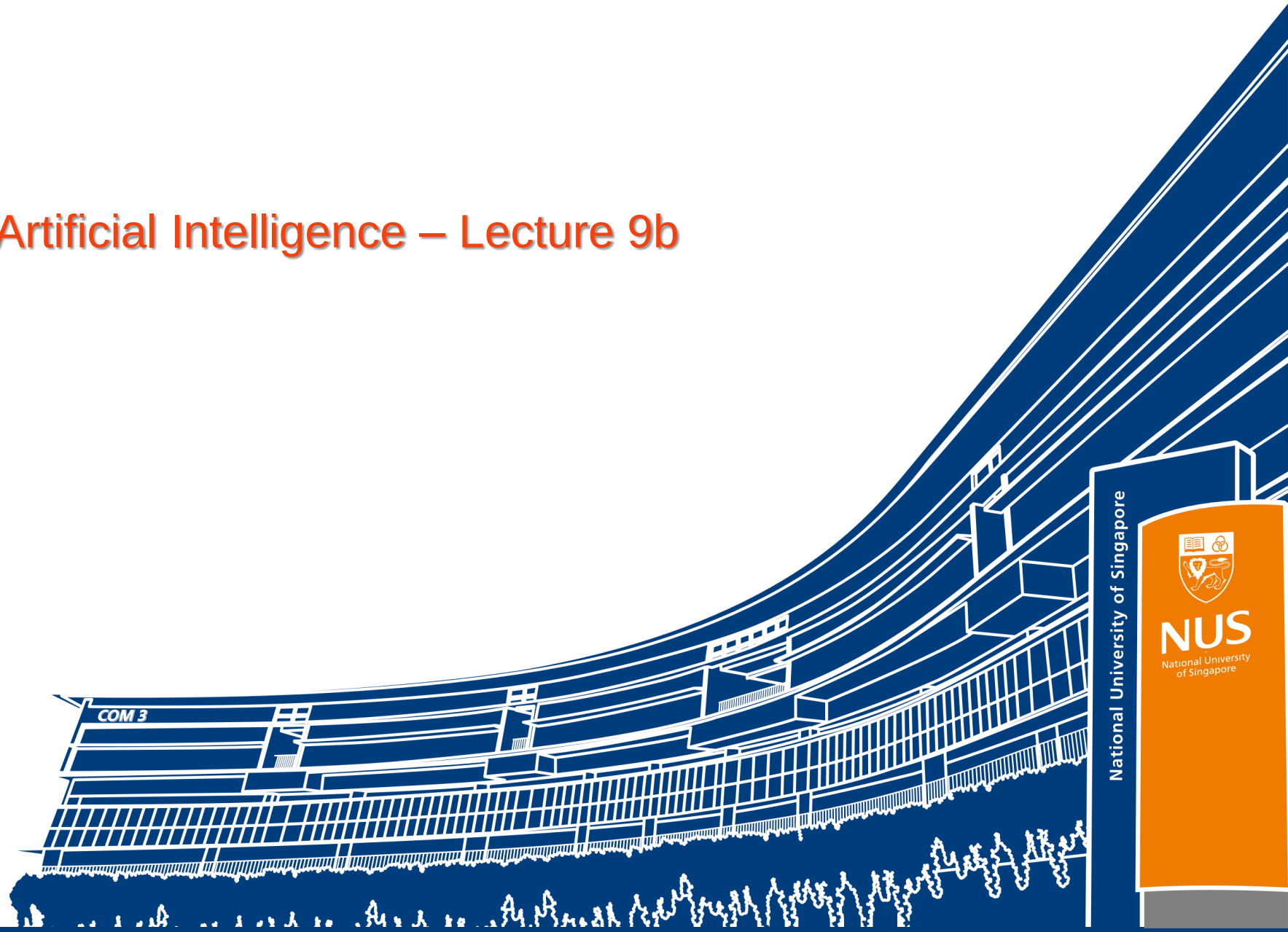
```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

construct a sentence relating to an action to take

- Inference algorithms ($\mathcal{A}$) assumes α is given and decides if $KB \models \alpha$
- When discussing soundness and completeness of $\mathcal{A}$, we consider which among any given/input α that will satisfy $KB \models \alpha$

Uncertainty I

CS3243: Introduction to Artificial Intelligence – Lecture 9b

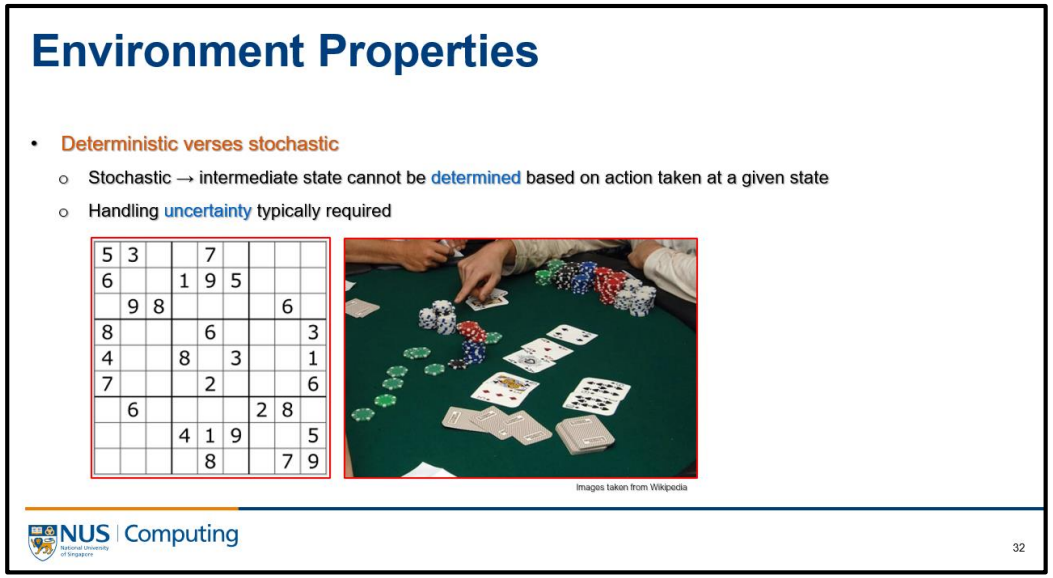


NUS | Computing
National University of Singapore

7 Handling Uncertainty

Recall Stochastic Environments

- When an environment is stochastic:
  - transition(state, action)
    → probable intermediate states
    - i.e., when taking an action, there is a probability distribution over successor states
  - What causes stochasticity?
    - Partial observability
    - Noisy sensors
    - Uncertainty in action outcomes
    - Complexity in modelling



From Lecture 1

Uncertainty: An Example

- Example
  - Let A_t denote an autonomous taxi agent's action, where:
    - A_t : leave for airport t minutes before a flight
    - Will A_t get me to the airport on time?
- Sources of uncertainty
  - Partial observability (e.g., road state, other drivers' plans)
  - Noisy sensors (e.g., traffic reports, fuel sensor)
  - Uncertainty in action outcomes (e.g., flat tire, accident)
  - Complexity in modelling and predicting traffic (e.g., congestion)
- Agent will probably either
  - Risk Falsehood – e.g., A_{25} will get me there on time
  - Reaches weaker conclusion – e.g., A_{25} will get me there on time if
    - There is no accident on the bridge
    - It does not rain
    - I do not get a flat tire

Under logic (certainty), you may require A_{1440} to reach the airport on time (i.e., stay overnight)

Better to consider the probability of being on time with more reasonable t ...

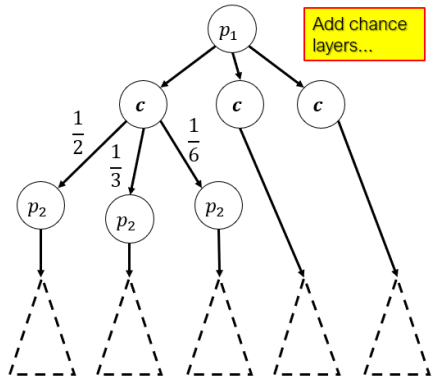
Handling Uncertainty

- Decision Theoretic Agents
  - Path Planning: Calculate highest probably of reaching goal (since cannot plan deterministically)
  - Game Playing: Calculate highest weighted utility (with weights based on probabilities)

Stochastic Games

- Many games have randomisation
  - Settlers of Catan
  - Poker
- How do we deal with uncertainty?
  - Can still use Minimax
  - Search tree is larger

Calculate expected value of a state (MUCH harder than deterministic games)



NUS | Computing

108

From Lecture 7

Requires the ability to calculate probabilities!

Since probabilities are often associated with Machine Learning, one may think that we are now concerned with Learning Agents. However, this is not the case

The focus in this last topic is on the Decision Theoretic Agent, which solves similar problems we have been addressing – i.e., planning (Problem-Solving Agent) and reasoning (Logical Agent).

8 Calculating Probabilities with Bayes' Theorem

Conditional Probabilities & Bayes Rule

- $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ assuming that $\Pr[B] > 0$

- Bayes rule: $\Pr[\, A \mid B \,] = \frac{\Pr[B|A] \Pr[A]}{\Pr[B]}$

Conditional Probabilities & Bayes Rule

- $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ assuming that $\Pr[B] > 0$

  From above, we have:
  Pr[A | B] = Pr[A ∧ B] / Pr[B] --- (1)
  Pr[B | A] = Pr[B ∧ A] / Pr[A] --- (2)

  Also, we have:
  Pr[A ∧ B] = Pr[B ∧ A] --- (3)

- Bayes rule: $\Pr[A \mid B] = \frac{\Pr[B|A] \Pr[A]}{\Pr[B]}$

Conditional Probabilities & Bayes Rule

- $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ assuming that $\Pr[B] > 0$

  From above, we have:
  Pr[A | B] = Pr[A ∧ B] / Pr[B] --- (1)
  Pr[B | A] = Pr[B ∧ A] / Pr[A] --- (2)

  Also, we have:
  Pr[A ∧ B] = Pr[B ∧ A] --- (3)

  From (2) and (3), we have:
  Pr[A ∧ B] = Pr[B | A] · Pr[A] --- (4)

- Bayes rule: $\Pr[\, A \mid B \,] = \frac{\Pr[B|A] \Pr[A]}{\Pr[B]}$

Conditional Probabilities & Bayes Rule

- $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ assuming that $\Pr[B] > 0$

From above, we have :
Pr[A | B] = Pr[A ∧ B] / Pr[B] --- (1)
Pr[B | A] = Pr[B ∧ A] / Pr[A] --- (2)

Also, we have:
Pr[A ∧ B] = Pr[B ∧ A] --- (3)

From (2) and (3), we have:
Pr[A ∧ B] = Pr[B | A] · Pr[A] --- (4)

And thus from (4) and (1), we have Bayes Rule:
P[A|B] = (P[B|A] · P[A]) / P[B]

- Bayes rule: $\Pr[\, A \mid B \,] = \frac{\Pr[B|A]\ \Pr[A]}{\Pr[B]}$

Next week, we will look at various applications of Bayes' Theorem to help define a Decision Theoretic Agent

Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/64539863216

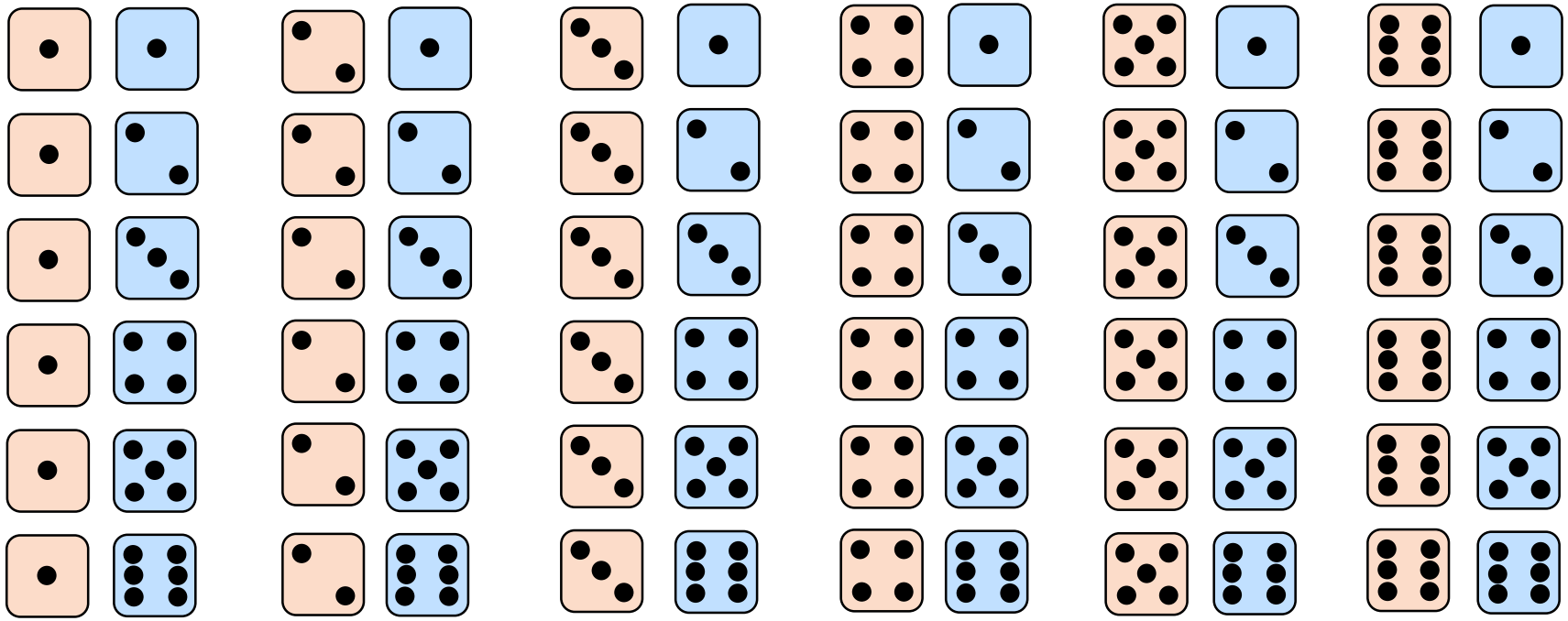
9 Appendix: Probability Fundamentals

Random Variables

- Random variable (X)
  - Quantifies an outcome of a random occurrence
    - e.g., outcome of a coin toss, die roll, or COVID-19 ART
- Domains (D_X)
  - Boolean: coin is either heads of tails (i.e., True or False)
  - Discrete: a die can have values {1, ..., 6}
- Events
  - Subsets of domains
    - e.g., Heads(X): coin flipped to heads
    - e.g., Even(X): die has value $\in$ {2,4,6}

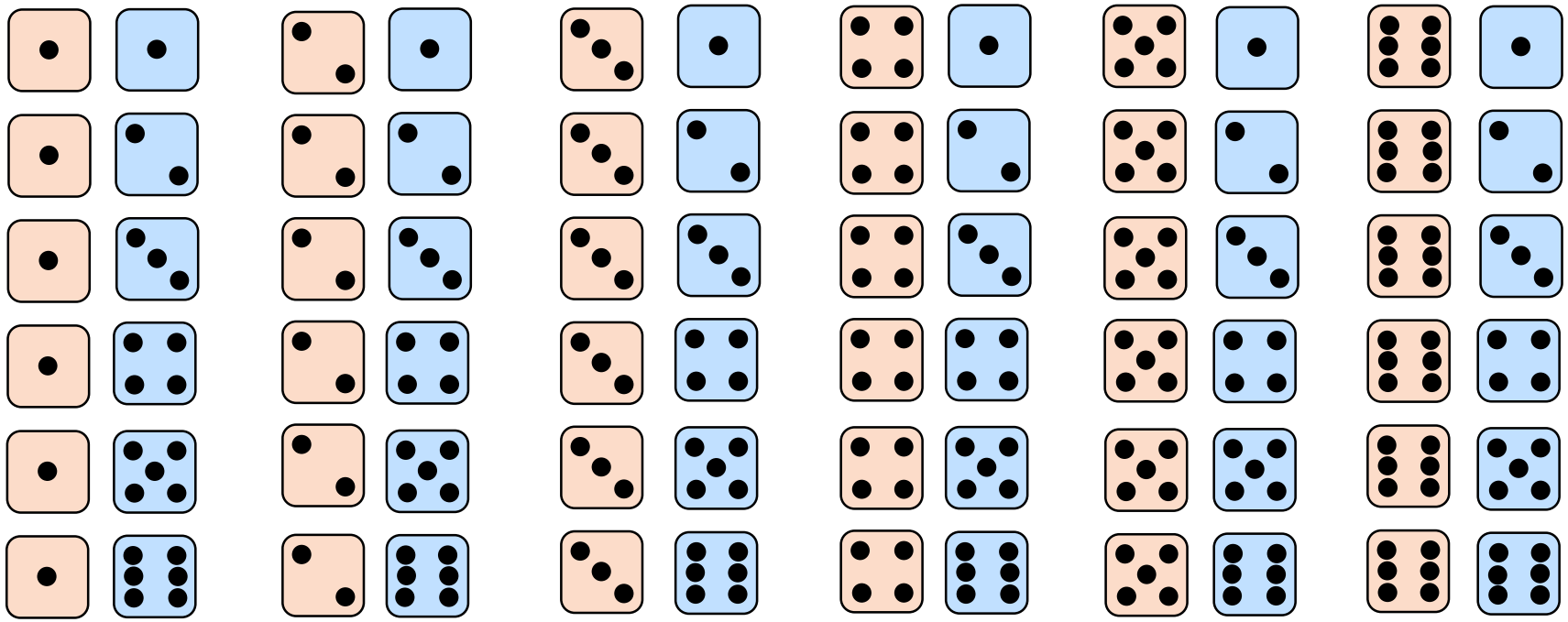
Events

- Atomic / singleton event (possible world)
  - An assignment of a value to each random variable
- Example – we roll two different dice



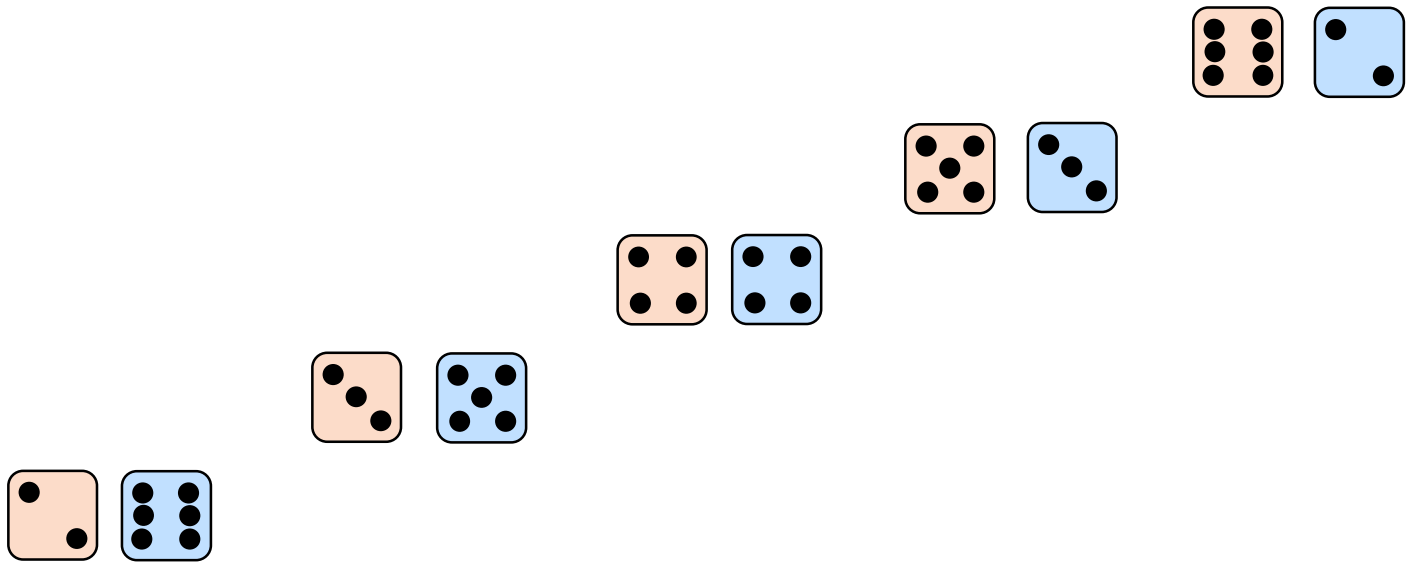
Events

- Yellow die: X_1
- Blue die: X_2



Events

- Yellow die: X_1
- Blue die: X_2
- Event: $X_1 + X_2 = 8$



Axioms of Probability

- Let X be a random variable with finite domain D_X
- A probability distribution over D_X assigns a value $p_X(v) \in [0,1]$ to every $v \in D_X$ such that

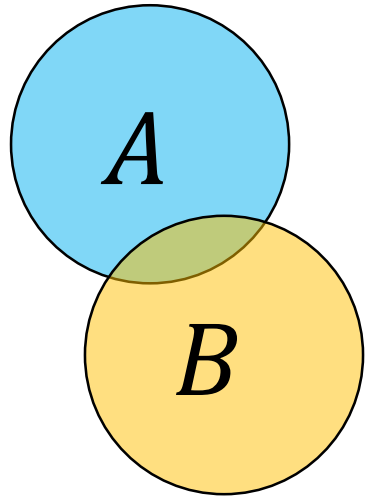
$$\sum_{v \in D_X} p_X(v) = 1$$

- For any event $A \subseteq D_X$, we have

$$\Pr[X \subseteq A] \equiv \Pr_X[A] = \sum_{v \in A} p_X(v)$$

- In particular

$$\Pr[A] + \Pr[B] = \Pr[A \cap B] + \Pr[A \cup B]$$



Joint Probability

- Given two random variables X and Y
  - The joint probability of an atomic event $(x, y) \in D_X \times D_Y$ is $p_{X,Y}(x, y) = \Pr[X = x \wedge Y = y]$
- In particular $p_X(x) = \sum_{y \in D_Y} p_{X,Y}(x, y)$
- Example

Income (in SGD) / AGE	15-24	25-34	35-44	45-54	55-64	65+
$< S\$2500$	0.062	0.051	0.037	0.019	0.015	0.039
$S\$2500 - S\$5000$	0.078	0.068	0.061	0.057	0.031	0.053
$> S\$5000$	0.015	0.051	0.094	0.119	0.111	0.039

Joint Probability

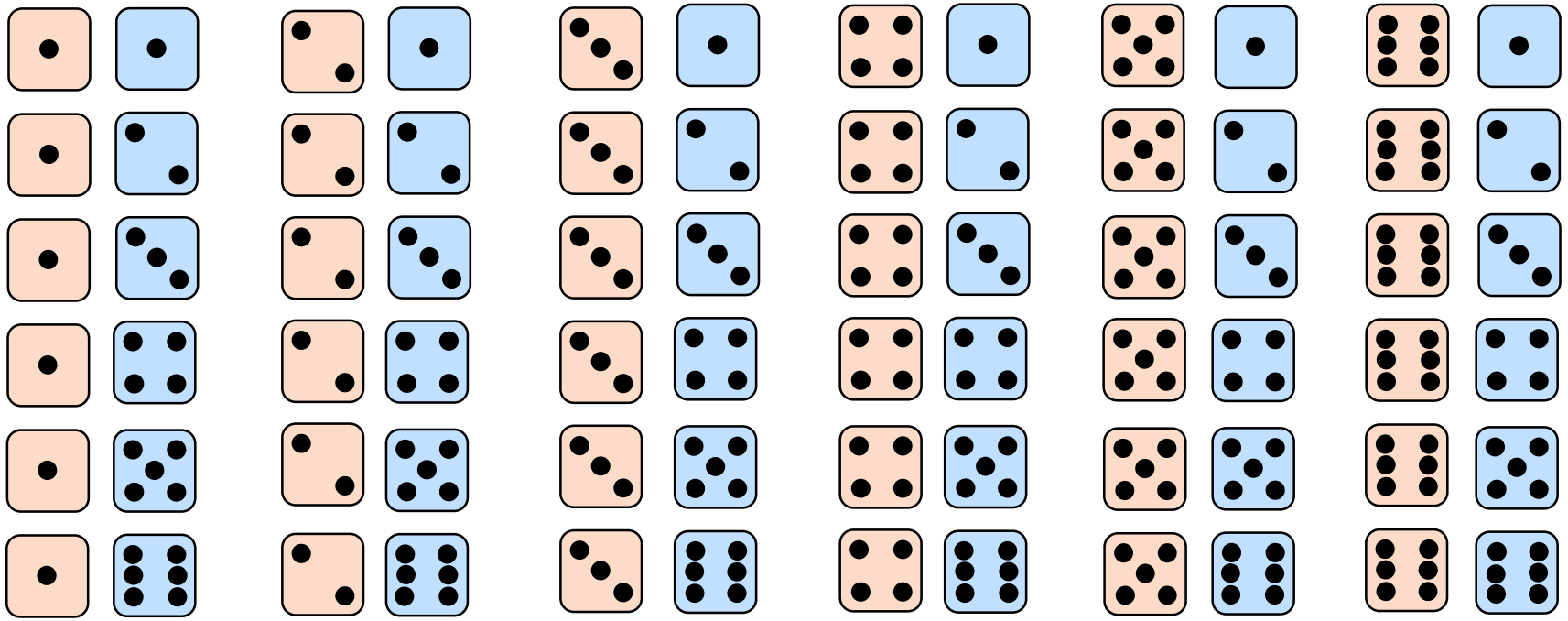
- Given two random variables X and Y
  - The joint probability of an atomic event $(x, y) \in D_X \times D_Y$ is $p_{X,Y}(x, y) = \Pr[X = x \wedge Y = y]$
- In particular $p_X(x) = \sum_{y \in D_Y} p_{X,Y}(x, y)$
- Example

Income (in SGD) / AGE	15-24	25-34	35-44	45-54	55-64	65+
$< S\$2500$	0.062	0.051	0.037	0.019	0.015	0.039
$S\$2500 - S\$5000$	0.078	0.068	0.061	0.057	0.031	0.053
$> S\$5000$	0.015	0.051	0.094	0.119	0.111	0.039

$$\Pr[Age = (25 - 34)] = 0.051 + 0.068 + 0.051 = 0.17$$

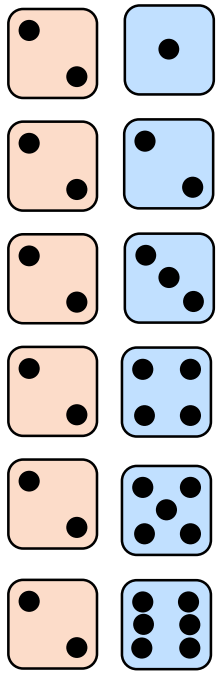
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example - rolling 2 dice; $\Pr[X_1 = 2] = \frac{?}{36}$



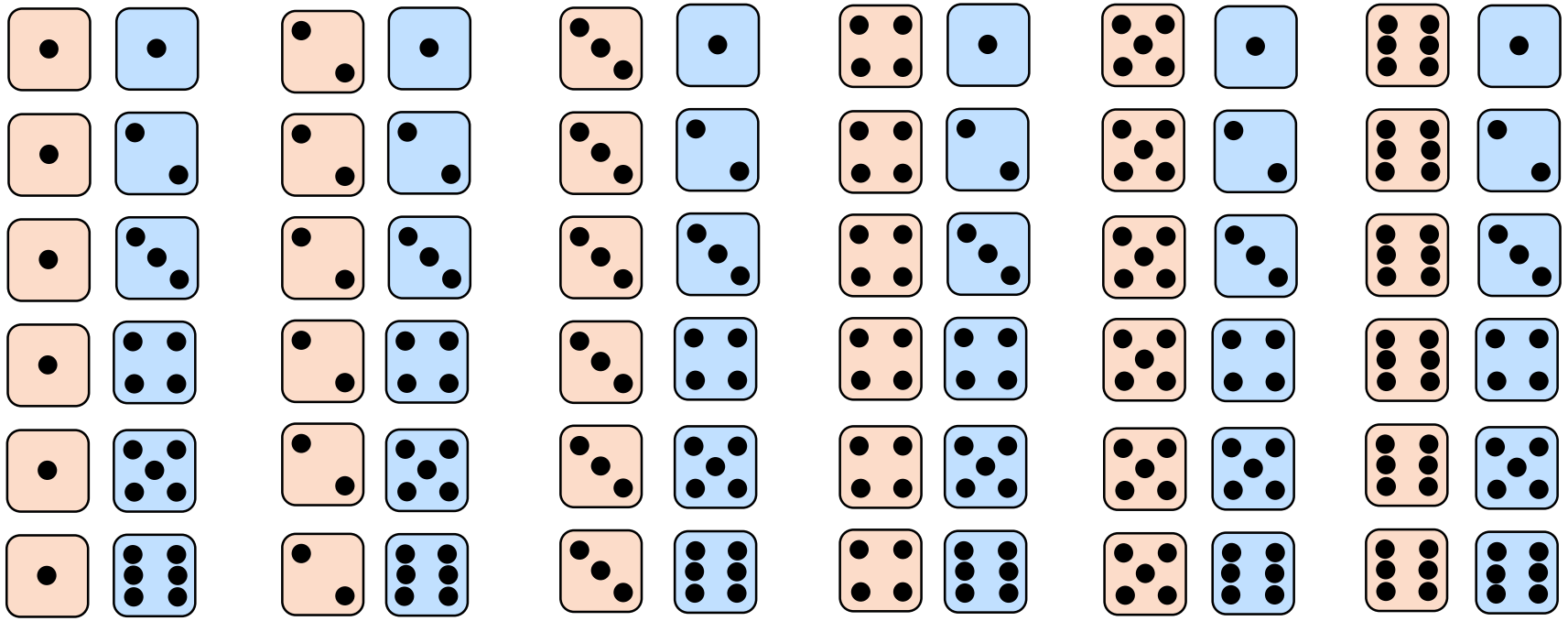
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example – rolling 2 dice; $\Pr[X_1 = 2] = \frac{6}{36}$



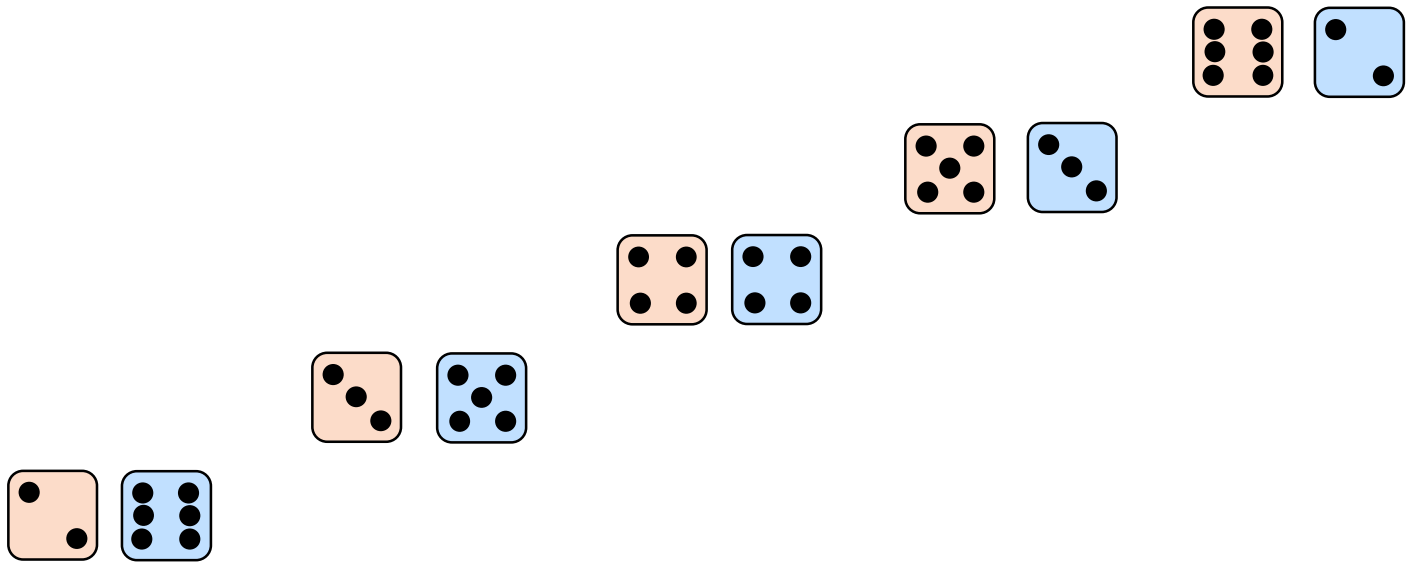
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example – rolling 2 dice; $\Pr[\, X_1 = 2 \mid X_1 + X_2 = 8 \,] = ?$



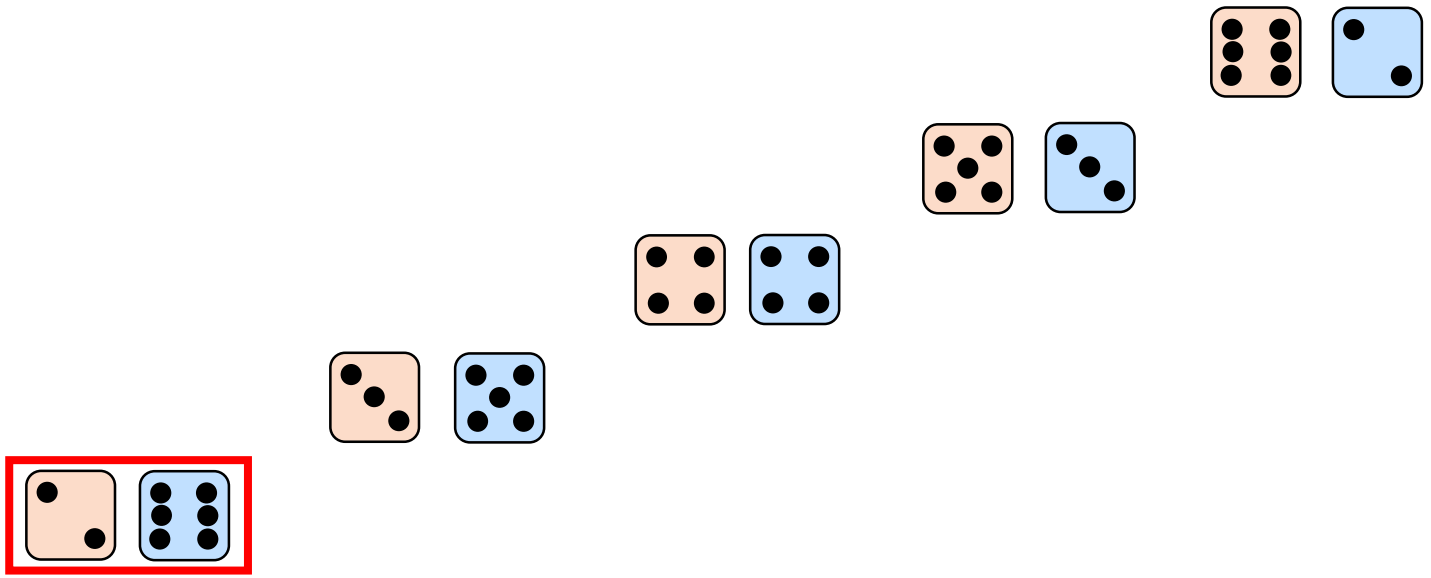
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example – rolling 2 dice; $\Pr[\, X_1 = 2 \mid X_1 + X_2 = 8 \,] = ?$



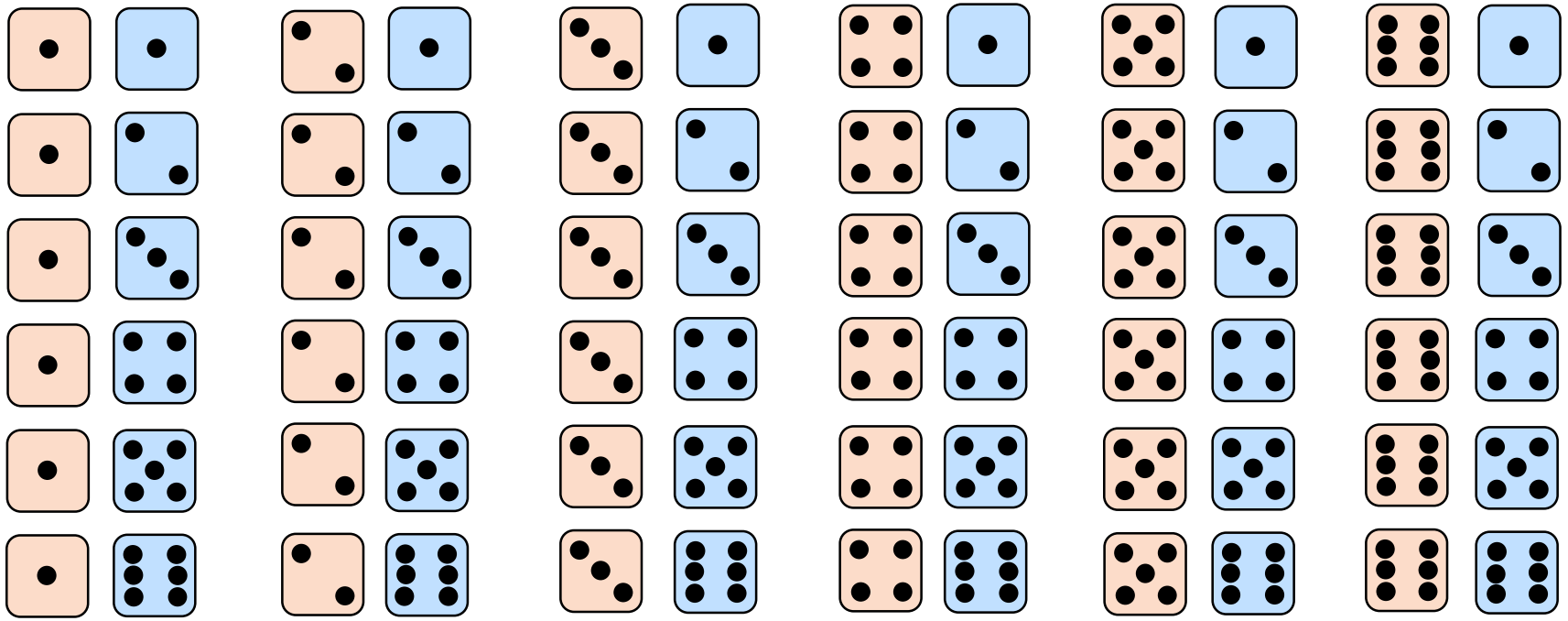
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example – rolling 2 dice; $\Pr[\, X_1 = 2 \mid X_1 + X_2 = 8 \,] = \frac{1}{5}$



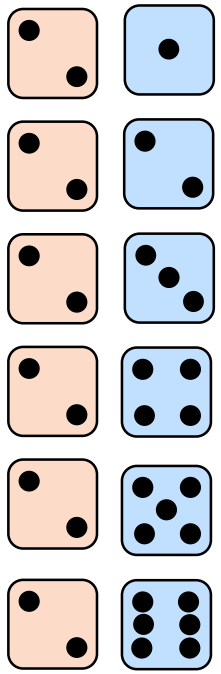
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example - rolling 2 dice; $\Pr[\, X_1 + X_2 = 8 \mid X_1 = 2 \,] = ?$



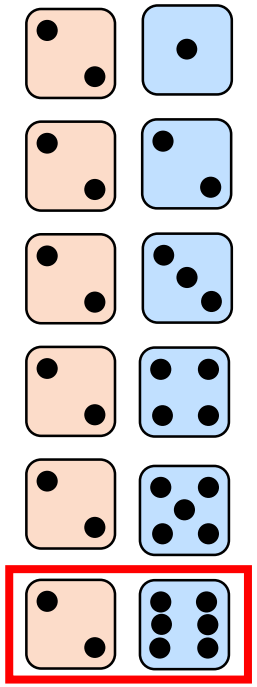
Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example – rolling 2 dice; $\Pr[\, X_1 + X_2 = 8 \mid X_1 = 2 \,] = ?$



Conditional Probability

- The probability that an event occurs, given that some other event occurs
- Example – rolling 2 dice; $\Pr[\, X_1 + X_2 = 8 \mid X_1 = 2 \,] = \frac{1}{6}$



Conditional Probabilities & Bayes Rule

- $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ assuming that $\Pr[B] > 0$

Note:
Pr[A | B] = Pr[A ∧ B] / Pr[B] --- (1)
Pr[B | A] = Pr[B ∧ A] / Pr[A] --- (2)

Also, we know:
Pr[A ∧ B] = Pr[B ∧ A] --- (3)

- Bayes rule: $\Pr[A \mid B] = \frac{\Pr[B|A]\,\Pr[A]}{\Pr[B]}$

From (2) and (3), we have:
Pr[A ∧ B] = Pr[B | A] · Pr[A] --- (4)

And thus from (4) and the definition above, we have Bayes Rule: P[A|B] = (P[B|A] · P[A]) / P[B]

Example: $\Pr[X_1 = 2 \mid X_1 + X_2 = 8] = ?$

Conditional Probabilities & Bayes Rule

- $\Pr[A \mid B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$ assuming that $\Pr[B] > 0$

Note:
Pr[A | B] = Pr[A ∧ B] / Pr[B] --- (1)
Pr[B | A] = Pr[B ∧ A] / Pr[A] --- (2)

Also, we know:
Pr[A ∧ B] = Pr[B ∧ A] --- (3)

- Bayes rule: $\Pr[A \mid B] = \frac{\Pr[B|A]\,\Pr[A]}{\Pr[B]}$

From (2) and (3), we have:
Pr[A ∧ B] = Pr[B | A] · Pr[A] --- (4)

And thus from (4) and the definition above, we have Bayes Rule: P[A|B] = (P[B|A] · P[A]) / P[B]

Example: $\Pr[X_1 = 2 \mid X_1 + X_2 = 8] = ?$

$$= \frac{\Pr[X_1+X_2=8|X_1=2]\cdot\Pr[X_1=2]}{\Pr[X_1+X_2=8]}$$

$$= \frac{1/6\cdot 1/6}{5/36} = \frac{1}{5}$$