COMP3702/7702 Artificial Intelligence
Week 5: Logic + Action Planning
R&N ch. 7.2-7.5, 10.2-10.3

Hanna Kurniawati
Topics until mid September

- What is AI.
- Search.
- Motion planning.
- Logic and its applications in action planning.
- Decision making under non-det. & adversarial uncertainty.
- Quantifying uncertainty.
- Intro to Bayesian Inference.
- Decision making under stochastic uncertainty.
Topics until mid September

- What is AI.
- Search.
- Motion planning.
- Logic and its applications in action planning.
- Decision making under non-det. & adversarial uncertainty.
- Quantifying uncertainty.
- Intro to Bayesian Inference.
- Decision making under stochastic uncertainty.
Topics until mid September

- What is AI.
- Search.
- Motion planning.
- Logic and its applications in action planning.
- Decision making under non-det. & adversarial uncertainty.
- Quantifying uncertainty.
- Intro to Bayesian Inference.
- Decision making under stochastic uncertainty.
Topics until mid September

- What is AI.
- Search.
- Motion planning.
- Logic and its applications in action planning.
- Decision making under non-det. & adversarial uncertainty + Quantifying uncertainty.
- Intro to Bayesian Inference.
- Decision making under stochastic uncertainty.
Agenda

- What is logic?
- Propositional logic.
- Action planning:
  - STRIPS representation.
  - Forward planning.
  - Planning graph.
  - Backward planning.
Agenda

- What is logic?
- Propositional logic.
- Action planning:
  - STRIPS representation.
  - Forward planning.
  - Planning graph.
  - Backward planning.
What is logic?

- A formal language to represent set of states.
  - A convenient abstraction for dealing with many states.
  - Instead of dealing with single states, it deals with set of states.

Where is it use?

- Action planning (since the first general purpose mobile robot, Shakey ‘69).
- Database, e.g., SQL (Structured Query Logic).
- Programming language, e.g., Prolog (Programming Logic).
A formal language

- Syntax
  - What expressions are legal.
  - Example:
    ```
    for (j = 0; j < 3; j++) { … }
    Coloring books sleep furiously.
    ```

- Semantics
  - Meaning of the legal expression.
Many types of logic

- Propositional logic.
- Predicate / first order logic.
- High order logic.
Propositional logic -- Syntax

- Atomic sentence.
  - An expression that is known to either be true or false.
  - Are these propositions?
    - There is life on Mars.
    - What is the distance between Mars and Earth?
    - $x + 2 = 2x$.
    - $x + 2 = 2x$ when $x = 1$.
    - $2x < 3x$.
    - This sentence is false.
  - Often represented with a symbol called propositional variable, e.g., P, Q.
Propositional logic -- Syntax

- Complex sentences
  - Constructed from atomic sentences and logical connectives: negation ($\neg$), and ($\land$), or ($\lor$), implication ($\rightarrow$), biconditional ($\iff$).
  - If $S$ is a sentence, $\neg S$ is a sentence (negation).
  - If $S_1$ and $S_2$ are sentences, $S_1 \land S_2$ is a sentence (conjunction).
  - If $S_1$ and $S_2$ are sentences, $S_1 \lor S_2$ is a sentence (disjunction).
  - If $S_1$ and $S_2$ are sentences, $S_1 \rightarrow S_2$ is a sentence (implication).
  - If $S_1$ and $S_2$ are sentences, $S_1 \iff S_2$ is a sentence (biconditional).
Propositional logic -- Semantics

- Truth values of a given sentence.
- Interpretation/model: Assignment of truth values to propositional variables.
- The truth value of a complex sentence can be derived from the truth values of logical connectives for the given interpretation.

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>~P</th>
<th>P &amp; Q</th>
<th>P \lor Q</th>
<th>P \rightarrow Q</th>
<th>P \leftrightarrow Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Usage

- Formulate information known as propositional logic sentences $\rightarrow$ knowledge base (R&N).
- Use knowledge base to deduce new information.
  - Does knowledge base entails a sentence $S$, i.e., does the sentence “knowledge base $\rightarrow S$” valid?
- Some terminologies:
  - A sentence is valid whenever the sentence is true for all models.
  - A sentence is satisfiable whenever there is at least one model such that the sentence is true.
  - A sentence is unsatisfiable whenever it is false for all models.
Example: Wumpus world

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>START</strong></td>
<td>Breeze</td>
<td>PIT</td>
<td>Breeze</td>
</tr>
<tr>
<td>2</td>
<td>Stench</td>
<td>Breeze</td>
<td>PIT</td>
<td>Breeze</td>
</tr>
<tr>
<td>3</td>
<td>Stench</td>
<td>Stench</td>
<td>PIT</td>
<td>Breeze</td>
</tr>
<tr>
<td>4</td>
<td>Stench</td>
<td>Breeze</td>
<td>PIT</td>
<td></td>
</tr>
</tbody>
</table>

Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter iff gold is in the same square
Suppose:

- $P_{x,y}$ is true if there’s a pit in $(x, y)$.
- $B_{x,y}$ is true if the agent perceives a breeze in $(x, y)$.

Sentences that are true based on the problem desc.:

- $S_1$: $\neg P_{1,1}$.
- $S_2$: $B_{1,1} \iff (P_{1,2} \lor P_{2,1})$.
- $S_3$: $B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})$.

Suppose the agent receive percepts that no breeze in $(1,1)$ and breeze in $(2,1)$. Additional true sentences:

- $S_4$: $\neg B_{1,1}$.
- $S_5$: $B_{2,1}$.
Question: Can we conclude there’s no pit in (1, 2)?
- Easy for us to “see”, but how to enable computers “see” it, too?
- Formally: Does the sentence
  \[ S_1 \land S_2 \land S_3 \land S_4 \land S_5 \implies \neg P_{1,2} \]
  valid?
- Two ways:
  - Model checking.
    - Check validity by checking all models.
  - Theorem proving
    - Check validity without checking all models.
(Simple) Model checking

- Enumerate the models.
  - All true/false values for $P_{1,1}$, $B_{1,1}$, $P_{1,2}$, $P_{2,1}$, $B_{2,1}$, $P_{2,2}$, $P_{3,1}$.
  - Check if $\neg P_{1,2}$ is true in all models where the knowledge base $(S_1 \wedge S_2 \wedge S_3 \wedge S_4 \wedge S_5)$ is true.

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
(Simple) Model checking

- **Sound.**
  - The result is correct.

- **Complete**
  - It always gives an answer.

- **Complexity:**
  - Time: \(O(2^n)\).
  - Space: \(O(n)\).
  - \(n\) is \#propositional variables.

- For various heuristics to speed-up, check out RN 7.6.
Question: Can we conclude there’s no pit in (1, 2)?

- Easy for us to “see”, but how to enable computers “see” it, too?
- Formally: Does the sentence
  \[ S_1 \land S_2 \land S_3 \land S_4 \land S_5 \rightarrow \neg P_{1,2} \]
  valid?

**Two ways:**

- Model checking.
  - Check validity by checking all models.
- Theorem proving
  - Check validity without checking all models.
Logical equivalences

\[(\alpha \land \beta) \equiv (\beta \land \alpha)\] \hspace{1cm} \text{commutativity of } \land

\[(\alpha \lor \beta) \equiv (\beta \lor \alpha)\] \hspace{1cm} \text{commutativity of } \lor

\[((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))\] \hspace{1cm} \text{associativity of } \land

\[((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))\] \hspace{1cm} \text{associativity of } \lor

\[\neg(\neg \alpha) \equiv \alpha\] \hspace{1cm} \text{double-negation elimination}

\[(\alpha \implies \beta) \equiv (\neg \beta \implies \neg \alpha)\] \hspace{1cm} \text{contraposition}

\[(\alpha \implies \beta) \equiv (\neg \alpha \lor \beta)\] \hspace{1cm} \text{implication elimination}

\[(\alpha \iff \beta) \equiv ((\alpha \implies \beta) \land (\beta \implies \alpha))\] \hspace{1cm} \text{biconditional elimination}

\[\neg(\alpha \land \beta) \equiv (\neg \alpha \lor \neg \beta)\] \hspace{1cm} \text{De Morgan}

\[\neg(\alpha \lor \beta) \equiv (\neg \alpha \land \neg \beta)\] \hspace{1cm} \text{De Morgan}

\[(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))\] \hspace{1cm} \text{distributivity of } \land \text{ over } \lor

\[(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))\] \hspace{1cm} \text{distributivity of } \lor \text{ over } \land

\[\alpha, \beta: \text{Sentence (atomic or complex).}\]
Inference rules

- Transformation for logical expressions.
- Modus ponens
  \[ \alpha \rightarrow \beta \\
  \alpha \\
  \underline{\beta} \]
- Modus tollens
  \[ \alpha \rightarrow \beta \\
  \sim \beta \\
  \underline{\sim \alpha} \]
- And-elimination
  \[ \alpha \land \beta \\
  \underline{\alpha} \]
Theorem proving – Natural deduction

- Use inference rules to deduce a sentence from the set of sentences in the knowledge base.

Example (p.251 R&N 3rd ed., slightly shorter):

- $S_6$: Biconditional elimination to $S_2$
  \[(B_{1,1} \rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \rightarrow B_{1,1}).\]

- $S_7$: And-elimination to $S_6$
  \[(P_{1,2} \lor P_{2,1}) \rightarrow B_{1,1}.\]

- $S_8$: Modus tollens on $S_4$ and $S_7$
  \[\neg(P_{1,2} \lor P_{2,1}).\]

- $S_9$: De Morgan to $S_8$:
  \[\neg P_{1,2} \land \neg P_{2,1}.\]

- Use search.
Theorem proving – Natural deduction

- **State space:**
  - All possible set of sentences.

- **Action space:**
  - All inference rules.

- **World dynamics:**
  - Apply the inference rule to all sentences that match the above the line part of the inference rule. And add the sentence that lies below the line of the inference rule to the state.

- **Initial state:**
  - Initial knowledge base.

- **Goal state:**
  - The state contains the sentence we’re trying to prove.
Theorem proving – Natural deduction

- Sound.
- May not be complete.
  - Depend on whether we can provide a complete list of inference rules.
  - When we can, the branching factor can be very high.
Resolution

- A single inference rule.
  \[
  \alpha \lor \beta \\
  \neg \beta \lor \gamma \\
  \alpha \lor \gamma
  \]
  \(\alpha, \beta\): Sentence (atomic or complex).

- But, the single inference rule is sound & complete only when applied to propositional logic sentences written in Conjunctive Normal Form (CNF).
Conjunctive Normal Form (CNF)

- Conjunctions of disjunctions.
- Example: \((\neg A \lor B) \land (C \lor D) \land (E \lor F)\).
- Some terminologies
  - Clause: A disjunctions of literals, e.g., \((\neg A \lor B)\).
  - Literals: variables or the negation of variables, e.g., \(\neg A\) and \(B\).
Converting to CNF

- Every sentence in propositional logic can be written in CNF.

Three steps conversion:
  - Eliminate arrows using definitions.
  - Drive in negations using De Morgan’s Laws.
  - Distribute OR over AND.
Example

- Convert \((A \lor B) \rightarrow (C \rightarrow D)\) to CNF
- Eliminate arrows:
  \(~(A \lor B) \lor (\sim C \lor D)\)
- Drive in negations:
  \((\sim A \land \sim B) \lor (\sim C \lor D)\)
- Distribute OR over AND:
  \((\sim A \lor \sim C \lor D) \land (\sim B \lor \sim C \lor D)\)
Theorem proving -- Resolution refutation

- Three steps:
  - Convert all sentences into CNF.
  - Negate the desired conclusion.
  - Apply resolution rule until
    - Derive false (a contradiction).
    - Can’t apply the rule anymore.

- Sound & complete (for propositional logic).
  - If we derive a contradiction, the conclusion follows from the axioms.
  - If we can’t apply any more, the conclusion cannot be proved from the axioms.
Example

- Knowledge base: \((P \lor Q) \land (P \rightarrow R) \land (Q \rightarrow R)\).
  - Does the knowledge base entails R?

- Knowledge base: \(P \land \neg P\)
  - Does the knowledge base entails R?
  - Any sentence is entailed by the knowledge base if there’s a contradiction in the knowledge base. Meaning: Brittle when there’s uncertainty.
Real world applications?

- IC design & verification:
  - Pentium FDIV (floating point division) bug in 1994.
  - Since then, AMD, Intel, etc. use automated theorem proving to verify that its floating point calculations are implemented correctly in their processors.

- Less serious applications (or not 😊): Cluedo game player.
Agenda

- What is logic?
- Propositional logic.

**Action planning:**
- STRIPS language.
- Forward planning.
- Backward planning.
- Planning graph.
What is action planning?

- Action planning is computational methods that enable an agent to choose actions and ordering relations to achieve the given goals.
- Action planning decides higher level strategy, motion planning decides lower level strategy.

Example:
- Goal: Ask a robot to stock fridge with food & beverages.
- Action planning: Create a list of what to buy, go to Coles/ Woolies/… in Toowong, buy the food & beverages that have been listed before, go home, put groceries in the fridge.
- Motion planning: Find a path to go from apartment to car park, find a path to go from home to car park of Coles/Woolies/… in Toowong, etc.
STRIPS Language

- **State:**
  - Propositional logic sentences, where the logical connectives used are only conjunctions.

- **Action:**
  - Pre-condition state.
  - Post-condition state.
  - Basically, delete and add propositions to the state.

- **Initial state.**

- **Goal state.**
Example: Vacuum robot

- A robot needs to vacuum 2 rooms, R1 & R2.
- Relevant propositions:
  - Robot in R1, Robot in R2.
  - R1 clean, R2 clean.
  - States: conjunctions of the above propositions.
Example: Vacuum robot

- Initial state:
  - Robot in R1 $\land$ R1 is clean.

- Goal state:
  - R1 is clean $\land$ R2 is clean.
Example: Vacuum robot

- **Action:** Right
  - **Pre-condition:** Robot in R1 ∧ R1 is clean.
  - **Post-condition:** Robot in R2 ∧ R1 is clean.
  - **Delete:** Robot in R1, **Add:** Robot in R2.
  - An action A is applicable to a state S if the propositions in its precondition are all in S.

- **Often, we write as In(Robot, R1), ...**
Left
- $P = \text{In}(\text{Robot}, R_2)$
- $D = \text{In}(\text{Robot}, R_2)$
- $A = \text{In}(\text{Robot}, R_1)$

Suck($R_1$)
- $P = \text{In}(\text{Robot}, R_1)$
- $D = \emptyset$ [empty set]
- $A = \text{Clean}(R_1)$

Suck($R_2$)
- $P = \text{In}(\text{Robot}, R_2)$
- $D = \emptyset$ [empty set]
- $A = \text{Clean}(R_2)$
Planning

- Basically: search.
  - Build a tree with initial state as root.
  - Expand root with various actions.
  - In general, only need much fewer branching than possible. How to choose the “right” branches fast?
- Backward planning.
- Heuristic: Planning graph.
Backward planning

- Start from the goal, trying to reach the initial state.
- An action is relevant to achieving a goal if at least one proposition in its add list matches a proposition in the goal state.
- How to generate successor?
Backward planning – “Successor”

Successor(state G, action A) {
    If any proposition of G is in A’s delete list then return False.
    Else
        G’ ← Precondition of A
        For every propositions SG of G do
            If SG is not in A’s add list then add SG to G’
        Return G’
}
Heuristic: Planning Graph

- Search space for a relaxed version of the problem.
  - Consider each proposition independently.
- Help find reachable space of a state fast.
- The graph:
  - Vertices: propositions or actions.
  - Alternating layers of state and actions.
Heuristic: Planning Graph

- Start with the state’s propositions. → $S_0$
- All actions whose preconditions appear in $S_0$ become an action. → $A_0$
- $S_1$ contains all propositions that were in $S_0$ or are contained in the add lists of the actions in $A_0$
- So, $S_1$ contains all propositions that may be true in the state reached after the first action
- $A_1$ contains all actions not already in $A_0$ whose preconditions appear in $S_1$, hence that may be executable in the state reached after executing the first action. Etc…
In(Robot,R_1) → Clean(R_1) → In(Robot,R_2) → Clean(R_2) → In(Robot,R_1) → Clean(R_1)

Right Suck(R_1) → Suck(R_2) → Left