Logical Agents: Propositional Logic

Chapter 7
Outline

Topics:

• Knowledge-based agents
• Example domain: The Wumpus World
• Logic in general
  • models and entailment
• Propositional (Boolean) logic
• Equivalence, validity, satisfiability
• Inference rules and theorem proving
  • forward chaining
  • backward chaining
  • resolution
### Knowledge bases

<table>
<thead>
<tr>
<th>Inference engine</th>
<th>domain-independent algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge base</td>
<td>domain-specific content</td>
</tr>
</tbody>
</table>

- **Knowledge base** = set of *sentences* in a *formal* language
- **Declarative** approach to building an agent (or other system).
  - Declarative: Sentences express assertions about the domain
- Knowledge base operations:
  - *Tell* it what it needs to know
  - *Ask* (itself?) what to do – *query*
  - Answers should follow from the contents of the KB
Agents can be viewed:

- at the *knowledge level*
  - i.e., *what they know*, regardless of how implemented
- at the *implementation level* (also called the *symbol level*)
  - i.e., data structures and algorithms that manipulate them

️ Compare: abstract data type vs. data structure used to implement an ADT.
A simple knowledge-based agent

Function \textbf{KB-Agent}(\textit{percept}) \textbf{returns} an action

static: \textit{KB}, a knowledge base
\textit{t}, a counter, initially 0, indicating time

\textbf{Tell}(\textit{KB}, \textbf{Make-Percept-Sentence}(\textit{percept}, \textit{t}))

\textit{action} \leftarrow \textbf{Ask}(\textit{KB}, \textbf{Make-Action-Query}(\textit{t}))

\textbf{Tell}(\textit{KB}, \textbf{Make-Action-Sentence}(\textit{action}, \textit{t}))

\textit{t} \leftarrow \textit{t} + 1

\textbf{return} \textit{action}
A simple knowledge-based agent

In the most general case, the agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden/implicit properties of the world
- Deduce appropriate actions
The Wumpus World

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>START</strong></td>
<td>Breeze</td>
<td>PIT</td>
<td>Breeze</td>
</tr>
<tr>
<td>2</td>
<td>Stench</td>
<td>Gold</td>
<td>Breeze</td>
</tr>
<tr>
<td>3</td>
<td>Breeze</td>
<td>PIT</td>
<td>Breeze</td>
</tr>
<tr>
<td>4</td>
<td>Stench</td>
<td></td>
<td>PIT</td>
</tr>
</tbody>
</table>
Wumpus World PEAS description

**Performance measure:** gold: +1000; death: -1000; -1 per step; -10 for using the arrow

**Environment:**

- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting kills wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square

**Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot

**Sensors:** Breeze, Glitter, Smell, Bump, Scream
Wumpus world characterisation

Observable: ??
Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: ??
Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified

Episodic: ??
Wumpus world characterisation

**Observable:** No – only *local* perception

**Deterministic:** Yes – outcomes exactly specified

**Episodic:** No – sequential at the level of actions

**Static:** ??
Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified
  
  Episodic: No – sequential at the level of actions

  Static: Yes – Wumpus and pits do not move

Discrete: ??
Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified
  
  Episodic: No – sequential at the level of actions
  
  Static: Yes – Wumpus and pits do not move

Discrete: Yes

Single-agent: ??
Wumpus world characterisation

Observable: No – only *local* perception

Deterministic: Yes – outcomes exactly specified

   Episodic: No – sequential at the level of actions

   Static: Yes – Wumpus and pits do not move

Discrete: Yes

Single-agent: Yes – Wumpus is essentially a natural feature
Exploring a wumpus world

Percept:
[Stench: No, Breeze: No, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:
[Stench: No, Breeze: Yes, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:
[Stench: No, Breeze: Yes, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:

[Stench: Yes, Breeze: No, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:
[Stench: Yes, Breeze: No, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:
[Stench: No, Breeze: No, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:

[Stench: No, Breeze: No, Glitter: No, Bump: No, Scream: No]
Exploring a wumpus world

Percept:
[Stench: Yes, Breeze: Yes, Glitter: Yes, Bump: No, Scream: No]
Tight spots

- Breeze in (1,2) and (2,1)
  \[\Rightarrow\] no safe actions
Tight spots

- Breeze in (1,2) and (2,1) → no safe actions
- If pits are uniformly distributed, (2,2) is more likely to have a pit than (1,3) + (3,1)
Tight spots

- Smell in (1,1) ⇒ cannot safely move

Can use a strategy of coercion:
- shoot straight ahead
- wumpus was there ⇒ dead ⇒ safe
- wumpus wasn't there ⇒ safe
Tight spots

- Smell in (1,1) ⇒ cannot safely move
- Can use a strategy of *coercion*:
  - shoot straight ahead
  - wumpus was there ⇒ dead ⇒ safe
  - wumpus wasn’t there ⇒ safe
Logic in the Wumpus World

- As the agent moves and carries out sensing actions, it performs *logical reasoning*.

  - E.g.: “If (1,3) or (2,2) contains a pit and (2,2) doesn’t contain a pit then (1,3) must contain a pit”.

- We’ll use logic to represent information about the wumpus world, and to reason about this world.
Logic in general

• A *logic* is a formal language for representing information such that conclusions can be drawn

• The *syntax* defines the sentences in the language

• The *semantics* define the “meaning” of sentences;
  • i.e., define *truth* of a *sentence* in a *world*

• E.g., in the language of arithmetic
  • $x + 2 \geq y$ is a sentence; $x^2 + y >$ is not a sentence
  • $x + 2 \geq y$ is true iff the number $x + 2$ is not less than $y$
  • $x + 2 \geq y$ is true in a world where $x = 7, \ y = 1$
  • $x + 2 \geq y$ is false in a world where $x = 0, \ y = 6$
Semantics: Entailment

- **Entailment** means that one thing follows from another:
  \[ KB \models \alpha \]
- Knowledge base \( KB \) **entails** sentence \( \alpha \) if and only if:
  - \( \alpha \) is true in all worlds where \( KB \) is true
  - Or: if \( KB \) is true then \( \alpha \) must be true.
- E.g., the KB containing “the Canucks won” entails “either the Canucks won or the Leafs won”
- E.g., \( x + y = 4 \) entails \( 4 = x + y \)
- Entailment is a relationship between sentences (i.e., **syntax**) that is based on **semantics**
- Note: Brains (arguably) process **syntax** (of some sort).
Semantics: Models

- Logicians typically think in terms of *models*, which are complete descriptions of a world, with respect to which truth can be evaluated.
- We say *m is a model of* a sentence *α* if *α* is true in *m*.
- *M(α)* is the set of all models of *α*.
- Thus *KB |= α* if and only if *M(KB) ⊆ M(α)*.
- E.g. *KB = Canucks won and Leafs won*.
  *α = Canucks won*.
Aside: Semantics

- Logic texts usually distinguish:
  - an *interpretation*, which is some possible world or complete state of affairs, from
  - a *model*, which is an interpretation that makes a specific sentence or set of sentences true.

- The text uses *model* in both senses (so don’t be confused if you’ve seen the terms interpretation/model from earlier courses).
  - And if you haven’t, ignore this slide!

- We’ll use the text’s terminology.
Entailment in the Wumpus World

Consider the situation where the agent detects nothing in [1,1], moves right, detects a breeze in [2,1]

- Consider possible models for just the ?’s, assuming only pits

With no information:
- 3 Boolean choices $\Rightarrow$ 8 possible models
Wumpus Models

Consider possible arrangements of pits in [1,2], [2,2], and [3,1], along with observations:
Wumpus Models

Models of the KB:

- $KB = \text{wumpus-world rules} + \text{observations}$
Wumpus Models

- \( KB = \) wumpus-world rules + observations
- \( \alpha_1 = "[1,2] \text{ is safe}" \), \( KB \models \alpha_1 \), proved by model checking
• $KB = \text{wumpus-world rules} + \text{observations}$
Wumpus Models: Another Example

- $KB = \text{wumpus-world rules} + \text{observations}$
- $\alpha_2 = \text{“[2,2] is safe”, } KB \not\models \alpha_2$
Inference

In the case of propositional logic, we can use entailment to derive conclusions by enumerating models.

- This is the usual method of computing *truth tables*
- I.e. can use entailment to do *inference*.
- In first order logic we generally can’t enumerate all models (since there may be infinitely many of them and they may have an infinite domain).
- An *inference procedure* is a (syntactic) procedure for deriving some formulas from others.
Inference

- Inference is a procedure for computing entailments.
- \( KB \vdash \alpha = \) sentence \( \alpha \) can be derived from \( KB \) by the inference procedure.
- Entailment says what things are implicitly true in a KB.
- Inference is used to \textit{compute} things that are implicitly true.
Inference

- Inference is a procedure for computing entailments.
- $KB \vdash \alpha = \text{sentence } \alpha$ can be derived from $KB$ by the inference procedure.
- Entailment says what things are implicitly true in a KB.
- Inference is used to compute things that are implicitly true.

Desiderata:

- **Soundness**: An inference procedure is sound if whenever $KB \vdash \alpha$, it is also true that $KB \models \alpha$.
- **Completeness**: An inference procedure is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash \alpha$. 
Propositional Logic: Syntax

- Propositional logic is a simple logic – illustrates basic ideas
- We first specify the proposition symbols or (atomic) sentences: $P_1$, $P_2$ etc.
- Then we define the language:
  If $S_1$ and $S_2$ are sentences then:
  - $\neg S_1$ is a sentence (negation)
  - $S_1 \land S_2$ is a sentence (conjunction)
  - $S_1 \lor S_2$ is a sentence (disjunction)
  - $S_1 \Rightarrow S_2$ is a sentence (implication)
  - $S_1 \equiv S_2$ is a sentence (biconditional)
Propositional Logic: Semantics

- Each model assigns true or false to each proposition symbol.
- E.g.: $P_{1,2} \leftarrow true$, $P_{2,2} \leftarrow true$, $P_{3,1} \leftarrow false$
  (With these symbols, 8 possible models, can be enumerated.)
- Rules for evaluating truth with respect to a model $m$:
  \[
  \neg S \text{ is true iff } S \text{ is false}
  \]
  \[
  S_1 \land S_2 \text{ is true iff } S_1 \text{ is true and } S_2 \text{ is true}
  \]
  \[
  S_1 \lor S_2 \text{ is true iff } S_1 \text{ is true or } S_2 \text{ is true}
  \]
  \[
  S_1 \rightarrow S_2 \text{ is true iff } S_1 \text{ is false or } S_2 \text{ is true}
  \]
  \[
  S_1 \equiv S_2 \text{ is true iff } S_1 \Rightarrow S_2 \text{ is true and } S_2 \Rightarrow S_1 \text{ is true}
  \]
- Simple recursive process evaluates an arbitrary sentence, e.g.,

  \[
  \neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) = true \land (false \lor true) = true \land true = true
  \]
## Truth Tables for Connectives

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( \neg P )</th>
<th>( P \land Q )</th>
<th>( P \lor Q )</th>
<th>( P \Rightarrow Q )</th>
<th>( P \Leftrightarrow Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Wumpus World Sentences

• Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
• Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.
• Information from sensors: $\neg P_{1,1}$, $\neg B_{1,1}$, $B_{2,1}$
• Also know: “pits cause breezes in adjacent squares”
Wumpus World Sentences

- Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
- Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.
- Information from sensors: $\neg P_{1,1}, \neg B_{1,1}, B_{2,1}$
- “A square is breezy if and only if there is an adjacent pit”
  \[
  B_{1,1} \equiv (P_{1,2} \lor P_{2,1})
  \]
  \[
  B_{2,1} \equiv (P_{1,1} \lor P_{2,2} \lor P_{3,1})
  \]
- Note: $B_{1,1}$ has no “internal structure” – think of it as a string.
- So must write one formula for each square.
Wumpus World Sentences

- Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
- Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.
- Information from sensors: $\neg P_{1,1}, \neg B_{1,1}, B_{2,1}$
- “A square is breezy if and only if there is an adjacent pit”
  
  \[
  B_{1,1} \equiv (P_{1,2} \lor P_{2,1}) \\
  B_{2,1} \equiv (P_{1,1} \lor P_{2,2} \lor P_{3,1})
  \]

  - Note: $B_{1,1}$ has no “internal structure” – think of it as a string.
  - So must write one formula for each square.

- Using logic can conclude $\neg P_{1,2}$ and $\neg P_{2,1}$ from $\neg B_{1,1}$.
- Note, if you wrote the above as:
  
  \[
  B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})
  \]
  
  (i.e. “A breeze implies a pit in an adjacent square”)
  
  you could not derive $\neg P_{1,2}$ and $\neg P_{2,1}$ from $\neg B_{1,1}$.

  🚫 Crucial to express all information
Wumpus World KB

For the part of the Wumpus world we’re looking at, let

\[ KB = \{ R_1, R_2, R_3, R_4, R_5 \} \]

where

\[ R_1 \text{ is } \neg P_{1,1} \]
\[ R_2 \text{ is } B_{1,1} \equiv (P_{1,2} \lor P_{2,1}) \]
\[ R_3 \text{ is } B_{2,1} \equiv (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \]
\[ R_4 \text{ is } \neg B_{1,1} \]
\[ R_5 \text{ is } B_{2,1} \]
### Truth Tables for Inference

<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>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
</tr>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
</tr>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
</tr>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
</tr>
</tbody>
</table>

- Enumerate rows (different assignments to symbols),
- For $KB \models \alpha$, if KB is true in row, check that $\alpha$ is too.
Inference by Enumeration

Function $\text{TT-Entails?}(KB, \alpha)$ returns true or false
inputs: $KB$, the knowledge base, a sentence in propositional logic
$\alpha$ the query, a sentence in propositional logic
symbols $\leftarrow$ a list of the proposition symbols in $KB$ and $\alpha$
return $\text{TT-Check-All}(KB, \alpha, \text{symbols}, [])$
Inference by Enumeration

Function $\text{TT-Check-All}(KB, \alpha, \text{symbols}, \text{model})$ returns true or false
  
  if $\text{Empty?}(\text{symbols})$ then
  
    if $\text{PL-True?}(KB, \text{model})$ then return $\text{PL-True?}(\alpha, \text{model})$
    
    else return true
  
  else do
  
    $P \leftarrow \text{First}(\text{symbols});$ rest $\leftarrow \text{Rest}(\text{symbols})$
    
    return $\text{TT-Check-All}(KB, \alpha, \text{rest}, \text{model} \cup \{P = \text{true}\})$ and
    
    $\text{TT-Check-All}(KB, \alpha, \text{rest}, \text{model} \cup \{P = \text{false}\})$
  
• Depth-first enumeration of all models
  
• Hence, sound and complete

• Algorithm is $O(2^n)$ for $n$ symbols; problem is $\text{co-NP-complete}$
Other Means of Computing Logical Inference

- We’ll briefly consider other means of computing entailments:
  - Resolution theorem proving
  - Specialised rule-based approaches
- But first, some more terminology
Logical Equivalence

- Two sentences are *logically equivalent* iff true in same models:
  \[ \alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha \]
- The following should be familiar:

\[
\begin{align*}
  (\alpha \land \beta) & \equiv (\beta \land \alpha) \\
  (\alpha \lor \beta) & \equiv (\beta \lor \alpha) \\
  ((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \\
  ((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \\
  \neg(\neg \alpha) & \equiv \alpha \\
  (\alpha \Rightarrow \beta) & \equiv (\neg \beta \Rightarrow \neg \alpha) \\
  (\alpha \Rightarrow \beta) & \equiv (\neg \alpha \lor \beta) \\
  (\alpha \equiv \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \\
  \neg(\neg \land \beta) & \equiv (\neg \alpha \lor \neg \beta) \\
  \neg(\neg \lor \beta) & \equiv (\neg \alpha \land \neg \beta) \\
  (\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \\
  (\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))
\end{align*}
\]
Validity and Satisfiability

- A sentence is **valid** if it is true in *all* models, e.g., \( A \lor \neg A \), \( A \Rightarrow A \), \( (A \land (A \Rightarrow B)) \Rightarrow B \)

- Validity is connected to inference via the **Deduction Theorem**: \( KB \models \alpha \) if and only if \( (KB \Rightarrow \alpha) \) is valid

- A sentence is **satisfiable** if it is true in *some* model, e.g., \( A \lor B \), \( C \)

- A sentence is **unsatisfiable** if it is true in *no* models, e.g., \( A \land \neg A \)

- Satisfiability is connected to inference via the following: \( KB \models \alpha \) if and only if \( (KB \land \neg \alpha) \) is unsatisfiable
  - I.e., prove \( \alpha \) by **reductio ad absurdum**
  - What often proves better for determining \( KB \models \alpha \) is to show that \( KB \land \neg \alpha \) is unsatisfiable.
Validity and Satisfiability

- A sentence is **valid** if it is true in **all** models, e.g., $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the **Deduction Theorem**:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is **satisfiable** if it is true in **some** model, e.g., $A \lor B$, $C$
- A sentence is **unsatisfiable** if it is true in **no** models, e.g., $A \land \neg A$
- Satisfiability is connected to inference via the following:
  $KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable
  - I.e., prove $\alpha$ by **reductio ad absurdum**
  - What often proves better for determining $KB \models \alpha$ is to show that $KB \land \neg \alpha$ is unsatisfiable.
Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models,
  e.g., $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the *Deduction Theorem*:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is *satisfiable* if it is true in *some* model
  e.g., $A \lor B$, $C$
- What often proves better for determining $KB \models \alpha$ is to show that $KB \land \neg \alpha$ is unsatisfiable.
Validity and Satisfiability

- A sentence is *valid* if it is true in *all* models, e.g., $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the *Deduction Theorem*: $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is *satisfiable* if it is true in *some* model e.g., $A \lor B$, $C$
- A sentence is *unsatisfiable* if it is true in *no* models e.g., $A \land \neg A$
Validity and Satisfiability

- A sentence is **valid** if it is true in **all** models,
  e.g., $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the **Deduction Theorem**:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is **satisfiable** if it is true in **some** model
  e.g., $A \lor B$, $C$
- A sentence is **unsatisfiable** if it is true in **no** models
  e.g., $A \land \neg A$
- Satisfiability is connected to inference via the following:
  $KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable
  - i.e., prove $\alpha$ by **reductio ad absurdum**
Validity and Satisfiability

- A sentence is valid if it is true in all models, e.g., $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the Deduction Theorem: $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is satisfiable if it is true in some model e.g., $A \lor B$, $C$
- A sentence is unsatisfiable if it is true in no models e.g., $A \land \neg A$
- Satisfiability is connected to inference via the following: $KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable
  - i.e., prove $\alpha$ by reductio ad absurdum
- What often proves better for determining $KB \models \alpha$ is to show that $KB \land \neg \alpha$ is unsatisfiable.
General Propositional Inference: Resolution

Resolution is a rule of inference defined for *Conjunctive Normal Form* (CNF)

- **CNF:** conjunction of *disjunctions* of *literals*
- A *clause* is a *disjunctions* of *literals*.
- E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$.
  - Write as: $(A \lor \neg B), (B \lor \neg C \lor \neg D)$
Resolution

- **Resolution** inference rule:

\[
\begin{align*}
\ell_1 \lor \cdots \lor \ell_k, & \quad m_1 \lor \cdots \lor m_n \\
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\end{align*}
\]

where \( \ell_i \) and \( m_j \) are complementary literals. (I.e. \( \ell_i \equiv \neg m_j \).)

- E.g., \( P_{1,3} \lor P_{2,2}, \neg P_{2,2} \)
  
  \[
  \frac{P_{1,3}}{P_{1,3}}
  \]
Resolution

- **Resolution** inference rule:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\]

where \(\ell_i\) and \(m_j\) are complementary literals. (I.e. \(\ell_i \equiv \neg m_j\).)

- E.g., \[ P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2} \]

\[ P_{1,3} \]

- If you can derive the “empty clause” from a set of clauses \(C\), then \(C\) is unsatisfiable.
  - E.g. \{\(A, \neg A \lor B, \neg B\}\} is unsatisfiable.
Resolution

- **Resolution** inference rule:

\[
\frac{\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n}{\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n}
\]

where \( \ell_i \) and \( m_j \) are complementary literals. (i.e. \( \ell_i \equiv \neg m_j \).)

- E.g., \( P_{1,3} \lor P_{2,2}, \neg P_{2,2} \)

\[
P_{1,3}
\]

- If you can derive the “empty clause” from a set of clauses \( C \), then \( C \) is unsatisfiable.
  
  - E.g. \( \{A, \neg A \lor B, \neg B\} \) is unsatisfiable.

- Resolution is sound and complete for propositional logic

  - I.e. \( KB \models \alpha \) iff
    
    \( KB \land \neg \alpha \) is unsatisfiable iff
    
    the empty clause can be obtained from \( KB \land \neg \alpha \)
    
    by resolution
Using resolution to compute entailments

To show whether $KB \models \alpha$, show instead that $KB \land \neg \alpha$ is unsatisfiable:

1. Convert $KB \land \neg \alpha$ into conjunctive normal form.
2. Use resolution to determine whether $KB \land \neg \alpha$ is unsatisfiable.
3. If so then $KB \models \alpha$; otherwise $KB \not\models \alpha$. 
Conversion to CNF

E.g.: $B_{1,1} \equiv (P_{1,2} \lor P_{2,1})$
Conversion to CNF

E.g.: \( B_{1,1} \equiv (P_{1,2} \lor P_{2,1}) \)

1. Eliminate \( \equiv \), replacing \( \alpha \equiv \beta \) with \((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)\).

\[(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})\]
Conversion to CNF

E.g.: $B_{1,1} \equiv (P_{1,2} \lor P_{2,1})$

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
   
   $$(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \lor \beta$.
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1})$$
Conversion to CNF

E.g.: $B_{1,1} \equiv (P_{1,2} \lor P_{2,1})$

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
   
   $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \lor \beta$.

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1})$

3. Move $\neg$ inwards using de Morgan’s rules and double-negation:

   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$
Conversion to CNF

E.g.: $B_{1,1} \equiv (P_{1,2} \lor P_{2,1})$

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
   
   
   \[
   (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})
   \]

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \lor \beta$.

   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1})
   \]

3. Move $\neg$ inwards using de Morgan’s rules and double-negation:

   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})
   \]

4. Apply distributivity law ($\lor$ over $\land$) and flatten:

   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})
   \]
Conversion to CNF

E.g.: $B_{1,1} \equiv (P_{1,2} \lor P_{2,1})$

1. Eliminate $\equiv$, replacing $\alpha \equiv \beta$ with $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
   
   
   $$(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \lor \beta$.
   
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1})$$

3. Move $\neg$ inwards using de Morgan’s rules and double-negation:

   
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$$

4. Apply distributivity law ($\lor$ over $\land$) and flatten:

   
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$$

For resolution, then write as

   
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}), (\neg P_{1,2} \lor B_{1,1}), (\neg P_{2,1} \lor B_{1,1})$$
Resolution Algorithm

Function PL-Resolution(KB, $\alpha$) returns true or false

inputs: KB, the knowledge base, a sentence in propositional logic
$\alpha$, the query, a sentence in propositional logic

\[
\text{clauses} \leftarrow \text{the set of clauses in } \text{CNF}(KB \land \neg\alpha) \\
\text{new} \leftarrow \{\} \\
\text{loop do} \\
\text{if clauses contains the empty clause then return true} \\
\text{if } C_i, C_j \text{ are resolvable clauses where} \\
\text{PL-Resolve}(C_i, C_j) \notin \text{clauses} \\
\text{then clauses } \leftarrow \text{clauses} \cup \text{PL-Resolve}(C_i, C_j) \\
\text{else return false}
\]

Note that the algorithm in the text is buggy
Resolution Example

• E.g.: \( KB = (B_{1,1} \equiv (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \),
  \( \alpha = \neg P_{1,2} \)

• Show \( KB \models \alpha \) by showing that \( KB \land \neg \alpha \) is unsatisfiable:
Resolution: Continued

There is a great deal that can be done to improve the basic algorithm:

- **Unit resolution**: propagate unit clauses (e.g. $\neg B_{1,1}$) as much as possible.
  - Note that this corresponds to the *minimum remaining values* heuristic in constraint satisfaction!

- Eliminate tautologies

- Eliminate redundant clauses

- Eliminate clauses with literal $\ell$ where the complement of $\ell$ doesn’t appear elsewhere.

- **Set of support**: Do resolutions on clauses with ancestor in $\neg \alpha$.
  - I.e. keep a focus on the goal.
Specialised Inference: Rule-Based Reasoning

• We consider a very useful, restricted case: *Horn Form*
  
  • KB = *conjunction* of *Horn clauses*

• Horn clause =
  
  • proposition symbol; or
  
  • A rule of the form:
    
    (conjunction of symbols) ⇒ symbol

• E.g., C, (B ⇒ A), (C ∧ D ⇒ B)

  Not: (¬B ⇒ A), (B ∨ A)
Horn clauses

Technically a Horn clause is a clause or disjunction of literals, with at most one positive literal.

- I.e. of form $A_0 \lor \neg A_1 \lor \cdots \lor \neg A_n$ or $\neg A_1 \lor \cdots \lor \neg A_n$
- These can be written: $A_1 \land \cdots \land A_n \Rightarrow A_0$ or $A_1 \land \cdots \land A_n \Rightarrow \bot$
- We won't bother with rules of the form $A_1 \land \cdots \land A_n \Rightarrow \bot$
  - Rules of this form are called integrity constraints.
  - They don't allow new facts to be derived, but rather rule out certain combinations of facts.
Reasoning with Horn clauses

- **Modus Ponens** (for Horn form): Complete for Horn KBs
  \[
  \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta
  \]

  - Can be used with forward chaining or backward chaining.
  - **Forward chaining:** Iteratively add new derived facts.
  - **Backward chaining:** From a query, work backwards through the rules to known facts.

These algorithms are very natural; forward chaining runs in linear time.
Reasoning with Horn clauses

- **Modus Ponens** (for Horn form): Complete for Horn KBs
  \[
  \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta
  \]
  \[
  \beta
  \]
- Can be used with **forward chaining** or **backward chaining**.
- **Forward chaining**: Iteratively add new derived facts
Reasoning with Horn clauses

- **Modus Ponens** (for Horn form): Complete for Horn KBs

  \[
  \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta
  \]

- Can be used with **forward chaining** or **backward chaining**.

- **Forward chaining**: Iteratively add new derived facts

- **Backward chaining**: From a query, work backwards through the rules to known facts.
Reasoning with Horn clauses

- **Modus Ponens** (for Horn form): Complete for Horn KBs
  \[ \alpha_1, \ldots, \alpha_n, \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta \]
- Can be used with *forward chaining* or *backward chaining*.
- **Forward chaining**: Iteratively add new derived facts
- **Backward chaining**: From a query, work backwards through the rules to known facts.
- These algorithms are very natural; forward chaining runs in *linear* time
Example

KB:

\[ P \Rightarrow Q, \]
\[ L \land M \Rightarrow P, \]
\[ B \land L \Rightarrow M, \]
\[ A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \]
\[ A, \]
\[ B \]
Forward chaining

Idea:

- Fire any rule whose premises are satisfied in the \( KB \),
- Add its conclusion to the \( KB \), until query is found
Forward chaining algorithm

Procedure:

\[ C := \{}; \]
\[
\text{repeat} \\
\quad \text{choose } r \in A \text{ such that } \\
\quad \quad r \text{ is } 'b_1 \land \cdots \land b_m \Rightarrow h' \\
\quad \quad b_i \in C \text{ for all } i, \text{ and} \\
\quad \quad h \not\in C; \\
\quad C := C \cup \{h\} \\
\quad \text{until no more choices}
\]
Forward chaining example

KB:
\[
\begin{align*}
P & \Rightarrow Q, \\
L \land M & \Rightarrow P, \\
B \land L & \Rightarrow M, \\
A \land P & \Rightarrow L, \\
A \land B & \Rightarrow L, \\
A & , \\
B & 
\end{align*}
\]

Query Q:

• From A and B, conclude L
• From L and B, conclude M
• From L and M, conclude P
• From P, conclude Q
Forward chaining example

KB:
\[ P \Rightarrow Q, \]
\[ L \land M \Rightarrow P, \]
\[ B \land L \Rightarrow M, \]
\[ A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \]
\[ A, \]
\[ B \]

Query Q:
- From A and B, conclude L
Forward chaining example

KB:

\[ P \Rightarrow Q, \]
\[ L \land M \Rightarrow P, \]
\[ B \land L \Rightarrow M, \]
\[ A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \]
\[ A, \]
\[ B \]

Query \( Q \):

- From \( A \) and \( B \), conclude \( L \)
- From \( L \) and \( B \), conclude \( M \)
Forward chaining example

KB:

\[
\begin{align*}
& P \Rightarrow Q, \\
& L \land M \Rightarrow P, \\
& B \land L \Rightarrow M, \\
& A \land P \Rightarrow L, \\
& A \land B \Rightarrow L, \\
& A, \\
& B
\end{align*}
\]

Query \( Q \):

- From \( A \) and \( B \), conclude \( L \)
- From \( L \) and \( B \), conclude \( M \)
- From \( L \) and \( M \), conclude \( P \)
Forward chaining example

KB:

\[ P \implies Q, \]
\[ L \land M \implies P, \]
\[ B \land L \implies M, \]
\[ A \land P \implies L, \]
\[ A \land B \implies L, \]
\[ A, \]
\[ B \]

Query \( Q \):

- From \( A \) and \( B \), conclude \( L \)
- From \( L \) and \( B \), conclude \( M \)
- From \( L \) and \( M \), conclude \( P \)
- From \( P \) conclude \( Q \)
Backward chaining

• We won't develop an algorithm for backward chaining, but will just consider it informally.

• Idea with backward chaining:
  Start from query $q$ and work backwards.

• To prove $q$ by BC:
  • check if $q$ is known already;
  • otherwise prove (by BC) all premises of some rule concluding $q$

• Avoid loops: Check if new subgoal is already on the goal stack

• Avoid repeated work: Check if new subgoal
  1. has already been proved true, or
  2. has already failed
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query \( Q \):

• Establish \( P \) as a subgoal.
• Can prove \( P \) by proving \( L \) and \( M \).
  • For \( M \):
    • Can prove \( M \) if we can prove \( B \) and \( L \).
      • \( B \) is known to be true.
      • \( L \) can be proven by proving \( A \) and \( B \).
        • \( A \) and \( B \) are known to be true.
  • For \( L \):
    • \( L \) can be proven by proving \( A \) and \( B \).
      • \( A \) and \( B \) are known to be true.
• \( L \) and \( M \) are true, thus \( P \) is true, thus \( Q \) is true.
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query Q:

- Establish \( P \) as a subgoal.
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query \( Q \):

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
Backward chaining example

KB:

\[ P \implies Q, \quad L \land M \implies P, \quad B \land L \implies M, \quad A \land P \implies L, \]
\[ A \land B \implies L, \quad A, \quad B \]

Query Q:

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
- For \( M \):
  - Can prove \( M \) if we can prove \( B \) and \( L \)
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query Q:

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
- For \( M \):
  - Can prove \( M \) if we can prove \( B \) and \( L \)
  - \( B \) is known to be true
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query Q:

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
- For \( M \):
  - Can prove \( M \) if we can prove \( B \) and \( L \)
  - \( B \) is known to be true
  - \( L \) can be proven by proving \( A \) and \( B \).
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query \( Q \):

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
- For \( M \):
  - Can prove \( M \) if we can prove \( B \) and \( L \)
  - \( B \) is known to be true
  - \( L \) can be proven by proving \( A \) and \( B \).
  - \( A \) and \( B \) are known to be true

\[ L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]
Backward chaining example

KB:

\[ P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L, \]
\[ A \land B \Rightarrow L, \quad A, \quad B \]

Query Q:

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
- For \( M \):
  - Can prove \( M \) if we can prove \( B \) and \( L \)
  - \( B \) is known to be true
  - \( L \) can be proven by proving \( A \) and \( B \).
  - \( A \) and \( B \) are known to be true
- For \( L \):
  - \( L \) can be proven by proving \( A \) and \( B \).
  - \( A \) and \( B \) are known to be true
Backward chaining example

KB:

\[
P \Rightarrow Q, \quad L \land M \Rightarrow P, \quad B \land L \Rightarrow M, \quad A \land P \Rightarrow L,
\]
\[
A \land B \Rightarrow L, \quad A, \quad B
\]

Query Q:

- Establish \( P \) as a subgoal.
- Can prove \( P \) by proving \( L \) and \( M \)
- For \( M \):
  - Can prove \( M \) if we can prove \( B \) and \( L \)
  - \( B \) is known to be true
  - \( L \) can be proven by proving \( A \) and \( B \).
  - \( A \) and \( B \) are known to be true
- For \( L \):
  - \( L \) can be proven by proving \( A \) and \( B \).
  - \( A \) and \( B \) are known to be true
- \( L \) and \( M \) are true, thus \( P \) is true, thus \( Q \) is true
Forward vs. backward chaining

- FC is *data-driven*, cf. automatic, unconscious processing,
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
  - Good for reactive agents
Forward vs. backward chaining

- FC is *data-driven*, cf. automatic, unconscious processing,
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
  - Good for reactive agents

- BC is *goal-driven*, appropriate for problem-solving,
  - E.g., Where are my keys? How do I get a job?
  - Complexity of BC can be *much less* than linear in size of KB
  - Can also sometimes be *exponential* in size of KB
  - Good for question-answering and explanation
Summary

• Logical agents apply *inference* to a *knowledge base* to derive new information and make decisions

• Basic concepts of logic:
  • *syntax*: formal structure of *sentences*
  • *semantics*: *truth* of sentences wrt *models*
  • *entailment*: necessary truth of one sentence given another
  • *inference*: deriving sentences from other sentences
  • *soundness*: derivations produce only entailed sentences
  • *completeness*: derivations can produce all entailed sentences
Summary (Continued)

- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Resolution is complete for propositional logic.
- Forward, backward chaining are complete for Horn clauses.
- Forward chaining is linear-time for Horn clauses.
- Propositional logic lacks expressive power