CS 4804: Intro to AI

Propositional Logic

Virginia Tech CS 4804 Fall 2021
Instructor: Yinlin Chen
Today’s Topics

• Knowledge-based agent
• Propositional logic
• Inference
Knowledge-Based Agents

• Agent has a central component: Knowledge base (KB)
  – KB: a set of *sentences* in a formal language
  – *Sentence*: Represents some assertion about the world using knowledge representation language

• Agent makes an action based on what it *knows* and its goals

• Declarative approach to build an agent
  – *TELL* it what it needs to know (or have it *Learn* the knowledge)
  – Then it can *ASK* itself what to do — answers should follow from the KB

• *Inference* algorithm can answer any *answerable* question

Knowledge base

Inference engine

- Domain-specific facts
- Generic code
Minesweeper

TELL:
• The number on a block shows the number of mines adjacent

ASK:
• Put a flag in a zone when you have confirmed that there is a mine (Action)

Goal
• You have to flag all the mines
function KB-AGENT(\textit{percept}) \textbf{returns} an action
persistent: $KB$, a knowledge base
$t$, a counter, initially 0, indicating time

\textbf{Tell}($KB$, MAKE-PERCEPT-SENTENCE($\textit{percept}, t$))
\textit{action} $\leftarrow$ \textbf{Ask}($KB$, MAKE-ACTION-QUERY($t$))
\textbf{Tell}($KB$, MAKE-ACTION-SENTENCE($\textit{action}, t$))
$t \leftarrow t + 1$

\textbf{return} $\textit{action}$

Inference
Logic

• Sentence:
  – Represents some assertion about the world using knowledge representation language
  – **Axiom:** a *sentence* that is taken to be *true*, to serve as a premise or starting point for further reasoning and arguments
  – Every *sentence* must be either *true* or *false* in each possible world (model)
  – New *sentence* is derived from old *sentence* through inference

• Knowledge representation language has syntax
  – \( X + Y = 4 \) is a well-formed sentence. \( 4xy+ \) is not.
Logic

- **Logic** has **syntax** and **semantics**
  - **Syntax**: What sentences are allowed?
  - **Semantics**:
    - What are the possible worlds?
    - Which sentences are true in which worlds? (i.e., definition of truth)

- Propositional symbol: P, Q, R
- Syntax: $P \lor (Q \land R)$
- **Semantics** defines the **truth** of each sentence with respect to each possible **world** (model)
  - $X + Y = 4$ is **true** if and only if (iff) in the **world** that $X = 2$ and $Y = 2$

- **Possible world**, a.k.a., **Model**
- A sentence $\alpha$ is true in model $m$.
  - $m$ **satisfies** $\alpha$
  - $m$ **is a model of** $\alpha$
- $M(\alpha)$: the set of all models of $\alpha$
Logic Example

• The sentence $x + y = 4$
• Possible world (model):
  – $\{x = 1, y = 3\}$
  – $\{x = 0, y = 4\}$
  – $\{x = 2, y = 2\}$
  – $\{x = 3, y = 4\}$
  – $\{x = 4, y = 2\}$
  – …
• $M(\alpha): \{{}\{x = 1, y = 3\}\}$,” $\{x = 0, y = 4\}$”, “$\{x = 2, y = 2\}$”, …..)
Logical Reasoning

• **Entailment:**
  - $\alpha \models \beta$: “sentence $\alpha$ entails sentence $\beta$”
  - $\alpha \models \beta$ iff in every model where $\alpha$ is true, $\beta$ is also true
  - $\alpha \models \beta$ iff $M(\alpha) \subseteq M(\beta)$
Logical Inference

• Entailment can be applied to derive conclusions
• Given KB with propositional logic sentences, check if sentence $\alpha$ is true
• Naive approach: Model-checking
  – Enumerates all possible models to check that $\alpha$ is true in all models in which KB is true
  – $M(KB) \subseteq M(\alpha)$
• Better approach: Theorem proving
  – Search for a sequence of proof steps (applications of inference rules) leading from $\alpha$ to $\beta$
Inference Algorithm

• New sentences can be derived from KB by an inference algorithm $i$

  $KB \vdash_i \alpha$

• **Sound or truth-preserving**: Inference algorithm that derives only entailed sentences
  – *Nothing but the truth*

• **Complete**: Inference algorithm can derive any sentence that is entailed
  – *The whole truth*
Propositional Logic - Syntax

- Proposition Symbol: P, Q, W_{1,2}, GoHokies...
- Logical connectives: ¬, ∧, ∨, ⇒, ⇔
- Complex sentences (Formula): ¬P, P ∧ Q, P ⇒ Q

Sentence → AtomicSentence | ComplexSentence

AtomicSentence → True | False | P | Q | R | ... 

ComplexSentence → (Sentence)
| ¬Sentence
| Sentence ∧ Sentence
| Sentence ∨ Sentence
| Sentence ⇒ Sentence
| Sentence ⇔ Sentence

Operator Precedence : ¬, ∧, ∨, ⇒, ⇔
## Constructing Logical Sentences - Python

<table>
<thead>
<tr>
<th>Operation</th>
<th>Book</th>
<th>Python Infix Input</th>
<th>Python Output</th>
<th>Python Expr Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>( \neg P )</td>
<td>(~P)</td>
<td>(~P)</td>
<td>Expr('~, P)</td>
</tr>
<tr>
<td>And</td>
<td>( P \land Q )</td>
<td>(P &amp; Q)</td>
<td>(P &amp; Q)</td>
<td>Expr('&amp;, P, Q)</td>
</tr>
<tr>
<td>Or</td>
<td>( P \lor Q )</td>
<td>(P \mid Q)</td>
<td>(P \mid Q)</td>
<td>Expr('</td>
</tr>
<tr>
<td>Inequality (Xor)</td>
<td>( P \neq Q )</td>
<td>(P \ ^\wedge Q)</td>
<td>(P ^ Q)</td>
<td>Expr('^', P, Q)</td>
</tr>
<tr>
<td>Implication</td>
<td>( P \rightarrow Q )</td>
<td>(P \mid '='\rightarrow\mid Q)</td>
<td>(P \rightarrow Q)</td>
<td>Expr('=&gt;', P, Q)</td>
</tr>
<tr>
<td>Reverse Implication</td>
<td>( Q \leftarrow P )</td>
<td>(Q \mid '='\leftarrow\mid P)</td>
<td>(Q \leftarrow P)</td>
<td>Expr('&lt;=', Q, P)</td>
</tr>
<tr>
<td>Equivalence</td>
<td>( P \leftrightarrow Q )</td>
<td>(P \mid '='\leftrightarrow\mid Q)</td>
<td>(P \leftrightarrow Q)</td>
<td>Expr('&lt;=&gt;', P, Q)</td>
</tr>
</tbody>
</table>
Propositional logic syntax

• Given: a set of proposition symbols \( \{ X_1, X_2, \ldots, X_n \} \)
  – (we often add True and False for convenience)
• \( X_i \) is a sentence
• If \( \alpha \) is a sentence then \( \neg \alpha \) is a sentence
• If \( \alpha \) and \( \beta \) are sentences then \( \alpha \land \beta \) is a sentence
• If \( \alpha \) and \( \beta \) are sentences then \( \alpha \lor \beta \) is a sentence
• If \( \alpha \) and \( \beta \) are sentences then \( \alpha \Rightarrow \beta \) is a sentence
• If \( \alpha \) and \( \beta \) are sentences then \( \alpha \Leftrightarrow \beta \) is a sentence
Propositional Logic - Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- $P \land Q$ is true iff $P$ is true and $Q$ is true in $m_1 = \{P = \text{true}, Q = \text{true}\}$
- $P \land Q$ is NOT true iff $P$ is false and $Q$ is true in $m_2 = \{P = \text{false}, Q = \text{true}\}$

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$\neg P$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \implies Q$</th>
<th>$P \iff Q$</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>
Propositional logic semantics

- Let $m$ be a model assigning true or false to \{x_1,x_2,..., x_n\}
- If $\alpha$ is a symbol then its truth value is given in $m$
- $\neg \alpha$ is true in $m$ iff $\alpha$ is false in $m$
- $\alpha \land \beta$ is true in $m$ iff $\alpha$ is true in $m$ and $\beta$ is true in $m$
- $\alpha \lor \beta$ is true in $m$ iff $\alpha$ is true in $m$ or $\beta$ is true in $m$
- $\alpha \Rightarrow \beta$ is true in $m$ iff $\alpha$ is false in $m$ or $\beta$ is true in $m$
- $\alpha \Leftrightarrow \beta$ is true in $m$ iff $\alpha \Rightarrow \beta$ is true in $m$ and $\beta \Rightarrow \alpha$ is true in $m$
Propositional logic semantics in code

function PL-TRUE?($\alpha$,model) returns true or false
    if $\alpha$ is a symbol then return Lookup($\alpha$, model)
    if Op($\alpha$) = $\neg$ then return not(PL-TRUE?(Arg1($\alpha$),model))
    if Op($\alpha$) = $\land$ then return and(PL-TRUE?(Arg1($\alpha$),model), PL-TRUE?(Arg2($\alpha$),model))
    etc.

(Sometimes called “recursion over syntax”)
Knowledge base

- M (Snow)
- M (Snow → Cold)
- M (Snow, Snow → Cold)
- M (Wet → Slippery)
- M (Rain → Wet)
- ....
$P_{x,y}$ is true if there is a pit in $[x, y]$.
$W_{x,y}$ is true if there is a wumpus in $[x, y]$, dead or alive.
$B_{x,y}$ is true if the agent perceives a breeze in $[x, y]$.
$S_{x,y}$ is true if the agent perceives a stench in $[x, y]$.

- There is no pit in $[1,1]$:
  \[ R_1 : \neg P_{1,1} \, . \]
- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:
  \[ R_2 : \quad B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \, . \]
  \[ R_3 : \quad B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \, . \]
- The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in Figure 7.3(b).
  \[ R_4 : \quad \neg B_{1,1} \, . \]
  \[ R_5 : \quad B_{2,1} \, . \]
Pacman’s knowledge base: Map

• Pacman knows where the walls are:
  – Wall_0,0 ∧ Wall_0,1 ∧ Wall_0,2 ∧ Wall_0,3 ∧ Wall_0,4 ∧ Wall_1,4 ∧ ...

• Pacman knows where the walls aren’t!
  – ¬Wall_1,1 ∧ ¬Wall_1,2 ∧ ¬Wall_1,3 ∧ ¬Wall_2,1 ∧ ¬Wall_2,2 ∧ ...
Theorem Proving

• Apply rules of inference directly to the sentences in KB to construct a proof of the desired sentence without consulting models

• Logical equivalences

• Validity

• Satisfiability
Logical equivalences

- $\alpha \equiv \beta$ iff $\alpha |\Rightarrow \beta$ and $\beta |\Rightarrow \alpha$

<table>
<thead>
<tr>
<th>Logical Equivalence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\alpha \land \beta) \equiv (\beta \land \alpha)$</td>
<td>commutativity of $\land$</td>
</tr>
<tr>
<td>$(\alpha \lor \beta) \equiv (\beta \lor \alpha)$</td>
<td>commutativity of $\lor$</td>
</tr>
<tr>
<td>$((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$</td>
<td>associativity of $\land$</td>
</tr>
<tr>
<td>$((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$</td>
<td>associativity of $\lor$</td>
</tr>
<tr>
<td>$\neg(\neg\alpha) \equiv \alpha$</td>
<td>double-negation elimination</td>
</tr>
<tr>
<td>$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$</td>
<td>contraposition</td>
</tr>
<tr>
<td>$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta)$</td>
<td>implication elimination</td>
</tr>
<tr>
<td>$(\alpha \iff \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$</td>
<td>biconditional elimination</td>
</tr>
<tr>
<td>$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$</td>
<td>De Morgan</td>
</tr>
<tr>
<td>$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$</td>
<td>De Morgan</td>
</tr>
<tr>
<td>$(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$</td>
<td>distributivity of $\land$ over $\lor$</td>
</tr>
<tr>
<td>$(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$</td>
<td>distributivity of $\lor$ over $\land$</td>
</tr>
</tbody>
</table>
Validity

• A sentence is valid if it is true in all models. E.g. \( P \lor \neg P \)
• Tautologies: valid sentences, they are necessarily true
• Every valid sentence is logically equivalent to True
• For any sentences \( \alpha \) and \( \beta \), \( \alpha \models \beta \) iff the sentence \( (\alpha \Rightarrow \beta) \) is valid (true in all models)
• If \( (\alpha \Rightarrow \beta) \) is true in all models, then \( \alpha \models \beta \)
Satisfiability (SAT)

- A sentence is satisfiable if it is true in some model
- SAT problem: the problem of determining the satisfiability of sentences in propositional logic (NP-complete problem)
- CSPs ask whether the constraints are satisfiable by some assignment.

<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>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>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>false</td>
<td>true</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>true</td>
</tr>
<tr>
<td>false</td>
<td>true</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>true</td>
</tr>
<tr>
<td>false</td>
<td>true</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>true</td>
</tr>
</tbody>
</table>

The table shows a truth assignment that satisfies the constraints.
Validity and Satisfiability

• $\alpha$ is valid iif $\neg \alpha$ is unsatisfiable (can’t find a single true in any model)
• $\alpha$ is satisfiable (find true in some model) iif $\neg \alpha$ is not valid
• $\alpha \models \beta$ iif the sentence $(\alpha \land \neg \beta)$ is unsatisfiable
Theorem Proving

• Apply **rules of inference** directly to the sentences in KB to **construct a proof** of the desired sentence **without consulting models**

• Generate **sound** inferences without the need for enumerating models
Inference Rules

• Modus Ponens: \[
\frac{\alpha \Rightarrow \beta, \; \alpha}{\beta}
\]
  - Whenever any sentences of the form \(\alpha \Rightarrow \beta\) and \(\alpha\) are given, sentence \(\beta\) can be inferred
  - \(\alpha \Rightarrow \beta\) and \(\alpha\) are in KB, new sentence \(\beta\) can be derived
  - \(\text{Rain} \Rightarrow \text{Wet}, \; \text{Rain}\)
    \[
    \frac{\text{Rain}}{\text{Wet}}
    \]
    Wet can be inferred

• And-Elimination: \[
\frac{\alpha \land \beta}{\alpha}
\]
  - From a conjunction, any of the conjuncts can be inferred
  - \(\text{Rain} \land \text{Wet}\)
    \[
    \frac{\text{Rain} \land \text{Wet}}{\text{Rain}}
    \]
    Wet can be inferred

• Monotonicity: if \(KB \models \alpha\) then \(KB \land \beta \models \alpha\)
Standard Logical Equivalences

\[(\alpha \land \beta) \equiv (\beta \land \alpha)\] commutativity of \(\land\)

\[(\alpha \lor \beta) \equiv (\beta \lor \alpha)\] commutativity of \(\lor\)

\[((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))\] associativity of \(\land\)

\[((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))\] associativity of \(\lor\)

\[\neg(\neg\alpha) \equiv \alpha\] double-negation elimination

\[\alpha \Rightarrow \beta \equiv (\neg\beta \Rightarrow \neg\alpha)\] contraposition

\[\alpha \Rightarrow \beta \equiv (\neg\alpha \lor \beta)\] implication elimination

\[\alpha \iff \beta \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))\] biconditional elimination

\[\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)\] De Morgan

\[\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)\] De Morgan

\[(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))\] distributivity of \(\land\) over \(\lor\)

\[(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))\] distributivity of \(\lor\) over \(\land\)
Example: Inference and Proof

KB

\[ R_1 : \neg P_{1,1} \]
\[ R_2 : B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \]
\[ R_3 : B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \]
\[ R_4 : \neg B_{1,1} \]
\[ R_5 : B_{2,1} \]

\[ R_2 : B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \]
\[ (\alpha \iff \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \] biconditional elimination
\[ R_6 : (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}) \]

And-Elimination

\[ R_7 : ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}) \]
\[ (\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha) \] contraposition
\[ R_8 : (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \lor P_{2,1})) \]
\[ R_4 : \neg B_{1,1} \]

Modus Ponens

\[ \frac{\alpha \Rightarrow \beta, \alpha}{\beta} \]

\[ R_9 : \neg(P_{1,2} \lor P_{2,1}) \]
\[ \neg(\alpha \lor \beta) \equiv (\neg \alpha \land \neg \beta) \] De Morgan
\[ R_{10} : \neg P_{1,2} \land \neg P_{2,1} \]

Goal

\[ \neg P_{1,2} \]
Resolution Inference Rule

- The resolution inference rule takes two implication sentences (of a particular form) and infers a new implication sentence.
- If KB contains A and B, we can infer $A \land B$.
- $A \lor B \lor C$
  - One of $A$, $B$, $C$ has a book and $B$ does not have a book, then $A$ or $C$ must have a book.
  - If we know $C$ does not have a book, the $A$ must have a book.
- $X_1 \lor X_2 \lor \ldots \lor X_m \lor \beta, Y_1 \lor Y_2 \lor \ldots \lor Y_n \lor \neg \beta$
  - A is a car or a bus, A is a bike or not a bus.
- $X_1 \lor X_2 \lor \ldots \lor X_m \lor Y_1 \lor Y_2 \lor \ldots \lor Y_n$
  - A is a car or a bike.
- Resolution is complete for propositional logic.
- Exponential time in the worst case.
Conjunctive Normal Form (CNF)

- Every logical sentence has a logically equivalent conjunctive normal form
- A sentence expressed as a conjunction of clauses is said to be in **CNF**
- We can formulate all the information contained in our knowledge base as one large conjunctive normal form
Example: CNF

\[ \text{KB} \]

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

1. Eliminate \( \iff \), replacing \( \alpha \iff \beta \) 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 \( \neg \alpha \lor \beta \):

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

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

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

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

\[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \]
Horn, Definite, and Goal clause

- **Definite clause**: a disjunction of literals of which exactly one literal is positive
- **Horn clause**: a disjunction of literals of which at most one literal is positive
- **Goal clause**: a disjunction of literals of which no literal is positive
- E.g. $A \lor B \lor C \lor D \lor E$
Horn clause

- Horn clause can be re-written as implications
  - E.g. \((\neg A \lor \neg B \lor C) \equiv (A \land B \Rightarrow C)\)

- Forward chaining applies Modus Ponens to generate new facts:
  - Given \(X_1 \land X_2 \land \ldots X_n \Rightarrow Y\) and \(X_1, X_2, \ldots, X_n\)
  - Infer \(Y\)

- Forward chaining keeps applying this rule, adding new facts, until nothing more can be added

- Requires KB to contain only definite clauses:
  - (Conjunction of symbols) \(\Rightarrow\) symbol; or
  - A single symbol (note that \(X\) is equivalent to True \(\Rightarrow X\))
Forward Chaining Algorithm

• From a set of inference rules, this algorithm goes through all possible sentences $f_1, \ldots, f_k$ and adds sentence $g$ to the knowledge base $KB$ if a matching rule exists.

• This process is repeated until no more additions can be made to $KB$.

• Forward chaining is sound and complete.

• Data-driven reasoning.
Forward Chaining Example: Proving Q

\[ P \implies Q \]
\[ L \land M \implies P \]
\[ B \land L \implies M \]
\[ A \land P \implies L \]
\[ A \land B \implies L \]
\[ A \]
\[ B \]
Backward-chaining

• How it works
  – Works backward from the query
    • If the query $q$ is true, done!
  – Finds implications in the KB whose conclusion is $q$
  – If all these implications can be proved true, the $q$ is true

• Goal-directed reasoning
• Touches only relevant facts
• Cost is much less
Backward-chaining Example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Reading and Next Class

• Logical Agents: 7.1 – 7.5
• First-order logic: 8