Logical Agents using Propositional Logic

Chapter 7
Knowledge bases

- Knowledge base = set of sentences in a formal language; here, Propositional Logic
  - List of things the agent ‘knows’
- Inference engine = processes this knowledge
- **Declarative** approach to building an agent (or other system):
  - Tell it what it needs to know
- Then it can **Ask** itself what to do - answers should follow from the KB
A simple knowledge-based agent

- It observes the world (via percepts)
- Makes an action based on percepts and knowledge
- It remembers its action
- Repeat

```
function KB-Agent(percept) returns an action
    static: 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
```
Example

KB: 1) [Goal is to enter room]
   2) [If [Goal is to enter room] and [robot is in front of room] and [door is closed], then [open door]]
   3) [If [Goal is to enter room] and [robot is in front of room] and [door is not closed], then [enter room]]

Percept:  [[Robot is in front of door] and [door is closed]]
How is this different than search?
CSP?
PL: Syntax & Semantics

• Syntax: Defines what a well-formed sentence is.
• Semantics: Defines the meaning of a sentence.

Me fail English? That’s unpossible!
Entailment

- **Entailment** means that one thing follows from another:
  \[ \text{KB} \models \alpha \]
- Knowledge base \( KB \) entails sentence \( \alpha \) if and only if \( \alpha \) is true in all worlds where \( KB \) is true
  - E.g., the KB containing “the Giants won” and “the Reds won” entails “Either the Giants won or the Reds won”
  - E.g., \( x+y = 4 \) entails \( 4 = x+y \)
  - Doesn’t necessarily go the other way;
    \( (P \land Q \models P) \) but it is not the case that \( (P \models P \land Q) \)
Inference

• $KB \vdash_i \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ by procedure } i$

• **Soundness:** $i$ is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
  – Everything it derives is correct

• **Completeness:** $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$
  – It is capable of deriving anything that can be derived from $KB$

• A procedure that derives $P$ from $(P \land Q)$ is sound, but not complete
  – Not applicable in handling $P \land (P \Rightarrow Q)$, for instance
Inference Example

KB:
1) [Not summer]
2) [In Seattle]
3) [If [In Seattle] and [Not summer], then [It is raining]]

Ask: Is [It is raining] true?
Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. P Q R
false true false

How many models are possible for n variables?

Rules for evaluating truth with respect to a model \( m \):

\[
\begin{align*}
\neg S & \quad \text{is true iff} \quad S \text{ is false} \\
S_1 \land S_2 & \quad \text{is true iff} \quad S_1 \text{ is true and } S_2 \text{ is true} \\
S_1 \lor S_2 & \quad \text{is true iff} \quad S_1 \text{ is true or } S_2 \text{ is true} \\
S_1 \implies S_2 & \quad \text{is true iff} \quad S_1 \text{ is false or } S_2 \text{ is true} \\
\text{i.e.,} \quad & \quad \text{is false iff} \quad S_1 \text{ is true and } S_2 \text{ is false} \\
S_1 \iff S_2 & \quad \text{is true iff} \quad S_1 \iff S_2 \text{ is true and } S_2 \iff S_1 \text{ is true}
\end{align*}
\]

Simple recursive process evaluates an arbitrary sentence, e.g.,

\[
\neg P \land (Q \lor R) = \text{true} \land (\text{true} \lor \text{false}) = \text{true} \land \text{true} = \text{true}
\]
Validity and satisfiability

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

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

What do each of these mean in terms of a truth table?
Inference, cont.

• So we’d like inference rules that are both sound and complete
  – Allow our agent to fully reason about its environment, given its knowledge

• None of the current rules is complete by itself
  – It’d really be nice to have a single rule that’s both sound and complete…
Resolution: One inference to rule them all

Conjunctive Normal Form (CNF)
conjunction of disjunctions of literals
clauses
E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

- **Resolution** inference rule (for CNF):

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

where \(\ell_i\) and \(m_j\) are complementary literals.
E.g., \(P_{1,3} \lor P_{2,2}, \neg P_{2,2}\)

\[
P_{1,3}
\]

- Resolution is sound and complete for propositional logic
Resolution

Soundness of resolution inference rule:

\[ \neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow l_i \]

\[ \neg m_j \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

\[ \neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

We just have to get everything in CNF…
Conversion to CNF

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

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

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 (\( \land \) over \( \lor \)) and flatten:
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})
   \]
Resolution algorithm

- Proof by contradiction, i.e., show $KB \land \neg \alpha$ unsatisfiable

```plaintext
function PL-RESOLUTION(KB, \alpha) returns true or false
    clauses ← the set of clauses in the CNF representation of $KB \land \neg \alpha$
    new ← \{
    loop do
        for each $C_i, C_j$ in clauses do
            resolvents ← PL-RESOLVE($C_i, C_j$)
            if resolvents contains the empty clause then return true
            new ← new \cup resolvents
        if new \subseteq clauses then return false
        clauses ← clauses \cup new
    end loop
    end function
```

2 Termination cases:
1) No new clauses are added by resolution; KB does not entail \alpha
2) Two clauses resolve to the ‘empty clause’; they cancel out.
   This happens when resolving a contradiction. KB does entail \alpha.
Forward chaining

• **Horn Form** (restricted)
  – KB = conjunction of Horn clauses
  – Horn clause =
    • proposition symbol; or
    • (conjunction of symbols) ⇒ symbol
  – E.g., C ∧ (B ⇒ A) ∧ (C ∧ D ⇒ B)
  – All symbols here are not negated
• **Modus Ponens** (for Horn Form): complete for Horn KBs
  \[ \alpha_1, \ldots, \alpha_n, \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta \]

• Can be used with **forward chaining**.
• These algorithms are very natural and run in linear time
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

```
function PL-FC-ENTAILS?(KB, q) returns true or false
local variables: count, a table, indexed by clause, initially the number of premises
               inferred, a table, indexed by symbol, each entry initially false
               agenda, a list of symbols, initially the symbols known to be true

  while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
      inferred[p] ← true
      for each Horn clause c in whose premise p appears do
        decrement count[c]
        if count[c] = 0 then do
          if HEAD[c] = q then return true
          PUSH(HEAD[c], agenda)
    return false
```

• Forward chaining is sound and complete for Horn KB
Forward chaining example