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

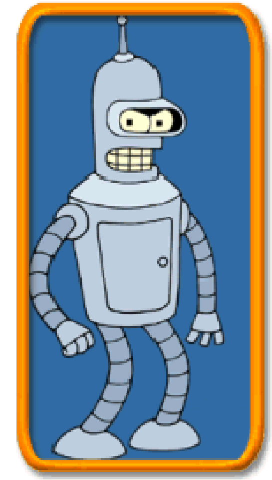
A simple knowledge-based agent

```
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
```

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

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]]

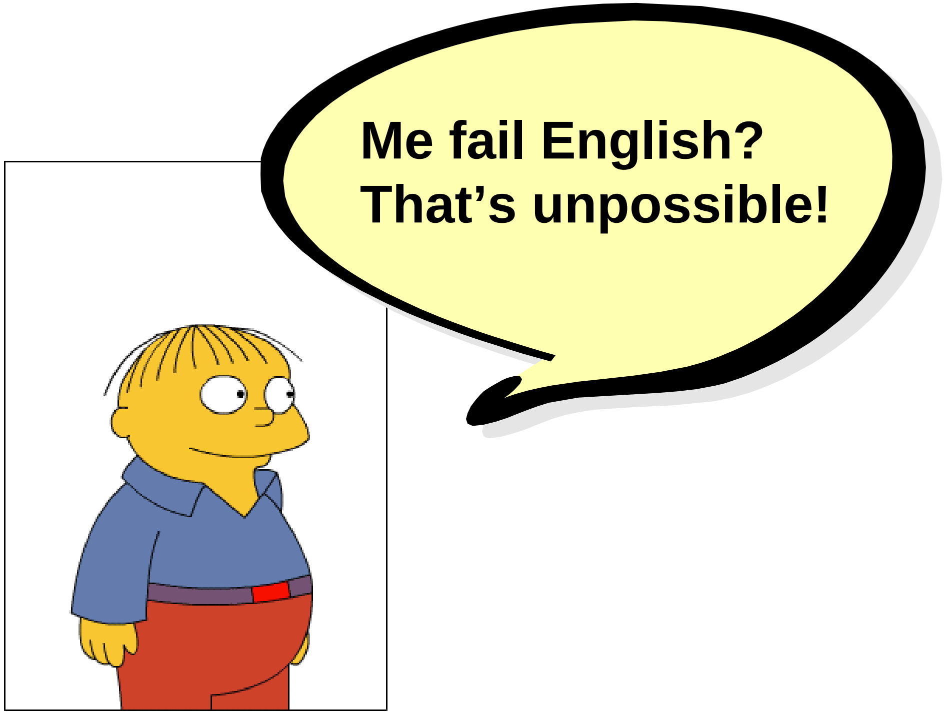
3) [If [Goal is to enter room] and [robot is in front of room] and [door is not closed], then [enter room]]

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.



Entailment

- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- Knowledge base *KB* entails sentence α if and only if α 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;

$$(P \wedge Q \models P) \text{ but it is not the case that } (P \models P \wedge Q)$$

Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from KB 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 $(P \wedge Q)$ is sound, but not complete
  - Not applicable in handling $P \wedge (P \Rightarrow Q)$, 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*:

$\neg S$	is true iff	S is false	
$S_1 \wedge S_2$	is true iff	S_1 is true and	S_2 is true
$S_1 \vee S_2$	is true iff	S_1 is true or	S_2 is true
$S_1 \Rightarrow S_2$	is true iff	S_1 is false or	S_2 is true
i.e.,	is false iff	S_1 is true and	S_2 is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and	$S_2 \Rightarrow S_1$ is true

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

$$\neg P \wedge (Q \vee R) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$$

Validity and satisfiability

A sentence is valid if it is true in all models,

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

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., $A \vee B$, C

A sentence is unsatisfiable if it is true in no models

e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ 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., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

- Resolution inference rule (for CNF):

$$\frac{\ell_i \vee \ldots \vee \ell_k, \qquad m_1 \vee \ldots \vee m_n}{\ell_i \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where ℓ_i and m_j are complementary literals.

E.g., $\dfrac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$

- Resolution is sound and complete for propositional logic

Resolution

Soundness of resolution inference rule:

$$\frac{\begin{array}{r}\neg(\ell_i \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k) \Rightarrow \ell_i \\ \neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\end{array}}{\neg(\ell_i \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

We just have to get everything in CNF…

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

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

3. Move $\neg$ inwards using de Morgan's rules and double-negation:□
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Apply distributivity law ($\wedge$ over $\vee$) and flatten:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution algorithm

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

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

2 Termination cases:

1) No new clauses are added by resolution; KB does not entail α
2) Two clauses resolve to the 'empty clause'; they cancel out. This happens when resolving a contradiction. KB does entail α.

Forward chaining

- Horn Form (restricted)
  - KB = conjunction of Horn clauses
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
  - All symbols here are not negated
- Modus Ponens (for Horn Form): complete for Horn KBs

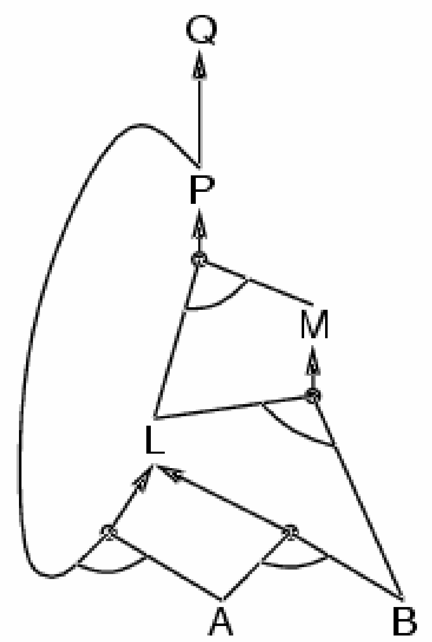
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- 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

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



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