Logical Agents

Chapter 6, AIMA2e Chapter 7

Outline

- Knowledge-based agents
- 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):

Tell it what it needs to know

Then it can Ask itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level

i.e., what they know, regardless of how implemented

Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

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

Sensors: Breeze, Glitter, Smell

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

Wumpus world characterization

Observable??
Wumpus world characterization

Observable? No—only local perception

Deterministic?

Episodic?

Static?

Wumpus world characterization

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 characterization

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 characterization

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

```
  +--------+
  |    A   |
  |        |
  |    OK  |
  +--------+

Exploring a wumpus world

```

```
  +--------+
  |    A   |
  |        |
  |    OK  |
  +--------+

Exploring a wumpus world

```

```
  +--------+
  |    A   |
  |        |
  |    OK  |
  +--------+

Exploring a wumpus world

```

```
  +--------+
  |    A   |
  |        |
  |    OK  |
  +--------+

Exploring a wumpus world

```

```
  +--------+
  |    A   |
  |        |
  |    OK  |
  +--------+

Exploring a wumpus world

```

```
  +--------+
  |    A   |
  |        |
  |    OK  |
  +--------+

Exploring a wumpus world

```
### Logic in general

**Logics** are formal languages for representing information such that conclusions can be drawn.

**Syntax** defines the sentences in the language.

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

E.g., the language of arithmetic:

- $x + 2 \geq y$ is a sentence.
- $2x + y >$ is not a sentence.
- $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number $y$.
- $x + 2 \geq y$ is true in a world where $x = 7$, $y = 1$.
- $x + 2 \geq y$ is false in a world where $x = 0$, $y = 6$.

### 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.

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 $4x + y$.

Entailment is a relationship between sentences (i.e., syntax) that is based on **semantics**.

Note: brains process **syntax** (of some sort).

### Other tight spots

- **Breeze** in (1,2) and (2,1) => no safe actions

  Assuming pits uniformly distributed,

  (2,2) has pit w/ prob 0.86, vs. 0.31.

- **Smell** in (1,1) => cannot move

  Can use a strategy of **coordon**:

  shoot straight ahead
  wumpus was there => dead => safe
  wumpus wasn’t there => safe

### Models

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated.

We say $m$ is a model of a sentence $\alpha$ if $\alpha$ is true in $m$.

$M(\alpha)$ is the set of all models of $\alpha$.

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$.

E.g., $KB$ = Giants won and Reds won

$\alpha$ = Giants won
Entailment in the wumpus world

Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for ?s assuming only pits
3 Boolean choices ⇒ 8 possible models

Wumpus models

KB = wumpus-world rules + observations

α₁ = [1, 2] is safe, KB ⊨ α₁, proved by model checking

Wumpus models

KB = wumpus-world rules + observations

Wumpus models

KB = wumpus-world rules + observations

Wumpus models

KB = wumpus-world rules + observations

α₂ = [2, 2] is safe, KB ⊭ α₂
Inference

$KB \vdash \alpha$ — sentence $\alpha$ can be derived from $KB$ by procedure $i$

Consequences of $KB$ are a haystack; $\alpha$ is a needle.

Entailment — needle in haystack; inference — finding it

Soundness: $i$ is sound if
whenever $KB \vdash \alpha$, it is also true that $KB \vdash \alpha$

Completeness: $i$ is complete if
whenever $KB \vdash \alpha$, it is also true that $KB \vdash \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough
to say almost anything of interest, and for which there exists a sound and
complete inference procedure.

That is, the procedure will answer any question whose answer follows from
what is known by the $KB$.

Truth tables for connectives

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$\neg P$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \Rightarrow Q$</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>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</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>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas.

The proposition symbols $P_1, P_2$ etc are sentences.

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, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional).

Wumpus world sentences

Let $P_{ij}$ be true if there is a pit in $[i, j]$.
Let $B_{ij}$ be true if there is a breeze in $[i, j]$.

$\neg P_{ij}$

$\neg B_{ij}$ $B_{ij}$

"Pits cause breezes in adjacent squares"
Truth tables for inference

<table>
<thead>
<tr>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
<th>P₅</th>
<th>P₆</th>
<th>KB</th>
<th>α</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>false</td>
</tr>
<tr>
<td>False</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</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>
</tr>
<tr>
<td>False</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</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>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>True</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>True</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>True</td>
<td>false</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>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>True</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>True</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Validity and satisfiability

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

- True, \(A \vee \neg A, A \Rightarrow A, \neg (A \wedge (A \Rightarrow B)) \Rightarrow B\)

Validity is connected to inference via the Deduction Theorem:

- \(KB \models \alpha\) if and only if \((KB \models \neg \alpha)\) is valid

A sentence is satisfiable if it is true in some model

- \(A \lor B, C\)

A sentence is unsatisfiable if it is true in no models

- \(A \land \neg A\)

Satisfiability is connected to inference via the following:

- \(KB \models \alpha\) if and only if \((KB \models \neg \alpha)\) is unsatisfiable

- i.e., prove \(\alpha\) by reductio ad absurdum

Inference by enumeration

Depth-first enumeration of all models is sound and complete

```text
function TT-ENTAILS?(KB, α) returns true or false
  symbols = a list of the proposition symbols in KB and α
  return TT-CHECK-ALL(KB, α, symbols, ( ))

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
  if Empty?(symbols) then
    if PL-TREE?(KB, model) then return PL-TREE?(α, model)
    else return true
  else do
    P = First(symbols); rest = Rest(symbols)
    return TT-CHECK-ALL(KB, α, rest, Expand(P, true, model) and
    TT-CHECK-ALL(KB, α, rest, Expand(P, false, model))

O(2^n) for n symbols; problem is co-NP-complete
```

Proof methods

Proof methods divide into (roughly) two kinds:

- **Application of inference rules**
  - Legitimate (sound) generation of new sentences from old
  - **Proof** - a sequence of inference rule applications
    - Can use inference rules as operators in a standard search alg.
    - Typically require translation of sentences into a normal form

- **Model checking**
  - truth table enumeration (always exponential in n)
  - Improved backtracking, e.g., Davis-Putnam-Logemann-Loveland
  - Heuristic search in model space (sound but incomplete)
    - e.g., mini-conflicts-like hill-climbing algorithms

Logical equivalence

Two sentences are logically equivalent if true in same models:

- \(\alpha \equiv \beta\) if and only if \(\alpha \models \beta\) and \(\beta \models \alpha\)

- \((\alpha \land \beta) = (\beta \land \alpha)\) commutativity of \(\land\)
- \((\alpha \lor \beta) = (\beta \lor \alpha)\) commutativity of \(\lor\)
- \((-\alpha) = (\beta \land \gamma)\) associativity of \(\land\)
- \((\alpha \land \beta) = (\beta \land \alpha)\) associativity of \(\lor\)
- \((\alpha \lor \beta) = (\beta \lor \alpha)\) double-negation elimination
- \((\alpha \Rightarrow \beta) = (\neg \beta \Rightarrow \neg \alpha)\) contraposition
- \((\alpha \Rightarrow \beta) = (\neg \alpha \lor \beta)\) implication elimination
- \((\alpha \equiv \beta) = (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)\) biconditional elimination
- \((\alpha \land (\beta \lor \gamma)) = ((\alpha \land \beta) \lor (\alpha \land \gamma))\) distributivity of \(\land\) over \(\lor\)
- \((\alpha \lor (\beta \land \gamma)) = ((\alpha \lor \beta) \land (\alpha \lor \gamma))\) distributivity of \(\lor\) over \(\land\)

Forward and backward chaining

**Horn Form (restricted)**

- \(KB\) = conjunction of Horn clauses
- Horn clause =
  - \(\diamond\) proposition symbol, or
  - \(\diamond\) conjunction of symbols \(\Rightarrow\) symbol
    - E.g., \(C \land (B \Rightarrow A) \land (A \land D \Rightarrow B)\)

- **Modus Ponens** (for Horn Form): complete for Horn KBs

- \(\alpha₁, \ldots, αₙ\) \(\alpha₁ \land \cdots \land αₙ \Rightarrow \beta\)

- Can be used with forward chaining or backward 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

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

Forward chaining algorithm

function PLFC-ENQUIRY(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
                P ← Ext[HEAD[c], agenda]
            end do
        end for
    end unless
end while

return false
Forward chaining example

Proof of completeness

FC derives every atomic sentence that is entailed by KB
1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model m, assigning true/false to symbols
3. Every clause in the original KB is true in m
   Proof: Suppose a clause \( a_1 \land \ldots \land a_k \Rightarrow b \) is false in m
   Then \( a_1 \land \ldots \land a_k \) is true in m and b is false in m.
   Therefore the algorithm has not reached a fixed point!
4. Hence m is a model of KB
5. If KB \( \models q \), q is true in every model of KB, including m

Backward chaining

Ideas: work backwards from the query q
   to prove q by BC,
   check if q is known already, or
   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

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

BC is goal-driven, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?
Complexity of BC can be much less than linear in size of KB
Resolution

Conjunctive Normal Form (CNF—universal)

Conjunction of disjunctions of literals

E.g., \((A \lor -B) \land (B \lor -C \lor -D)\)

Resolution: inference rule for CNF: complete for propositional logic

\[ \ell_1 \land \ldots \land \ell_k \lor m_1 \land \ldots \land m_k \]

where \(\ell_i\) and \(m_j\) are complementary literals. E.g.,

\[
P_{1,2} \lor P_{2,2} \quad -P_{2,2}
\]

\[
P_{1,3}
\]

Resolution is sound and complete for propositional logic

Conversion to CNF

1. Eliminate \(\iff\), replacing \(\alpha \iff \beta\) with \((\alpha \implies \beta) \land (\beta \implies \alpha)\).
2. Eliminate \(\Rightarrow\), replacing \(\alpha \Rightarrow \beta\) with \(-\alpha \lor \beta\).
3. Move \(-\) inwards using de Morgan’s rules and double-negation:
4. Apply distributivity law (\(\lor\) over \(\land\)) and flatten:

Resolution algorithm

Proof by contradiction, i.e., show \(KB \land \neg\alpha\) unsatisfiable

Function \(\text{PL-Resolution}(KB, \alpha)\) returns true or false

\[
\begin{align*}
\text{clauses} & := \text{the set of clauses in the CNF representation of } KB \land \neg\alpha \\
\text{new} & := \{\}
\end{align*}
\]

loop do

\[
\begin{align*}
\text{for each } C, C' \text{ in clauses do} & \\
\text{if } C \text{ contains the empty clause then return true} & \\
\text{end if} & \\
\text{if } C \text{ contains } \neg\alpha \text{ then return false} & \\
\text{end if} & \\
\text{if } C \text{ contains } \alpha \text{ then return true} & \\
\text{end if} & \\
\text{if } C \text{ contains } \neg\neg\alpha \text{ then return true} & \\
\text{end if} & \\
\text{if } C \text{ contains } \alpha \text{ then return false} & \\
\text{end if} & \\
\text{if } \alpha \text{ in clauses then return false} & \\
\text{end if} & \\
\text{if } \neg\neg\alpha \text{ in clauses then return true} & \\
\text{end if} & \\
\text{if } \text{new} \text{ clauses then return false} & \\
\text{end if} & \\
\text{clauses} & := \text{clauses} \cup \text{new}
\end{align*}
\]