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

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

Sensors
  Breeze, Glitter, Smell

A simple knowledge-based agent

```java
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())
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

Wumpus world characterization

Observable??
**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

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; \(x + 2 > y\) 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 \(4 = x + y\)

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

Note: brains process syntax (of some sort)

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 = \{\text{Giants won and Reds won}\}

\(\alpha = \text{Giants won}\)
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 $\Rightarrow$ 8 possible models

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

$\alpha_1 = \text{“[1,2] is safe”, } KB \models \alpha_1$, proved by model checking

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

$\alpha_2 = \text{“[2,2] is safe”, } KB \not\models \alpha_2$
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 $KB \vdash_1 \alpha$, it is also true that $KB \models \alpha$

Completeness: $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_1 \alpha$

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 $KB$.

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, $S_1 \iff S_2$ is a sentence (biconditional)

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 $[i,j]$.

$\neg P_{1,1}$
$\neg B_{1,1}$
$B_{2,1}$

“Pits cause breezes in adjacent squares”

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>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</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>true</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

Each model specifies true/false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{1,1}$

true true false

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model $m$:

- $\neg S$ is true iff $S$ is false
- $S_1 \land S_2$ is true iff $S_1$ is true and $S_2$ is true
- $S_1 \lor S_2$ 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
- $S_1 \iff S_2$ is true iff $S_1$ is true and $S_2$ is true

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

$\neg P_{1,2} \land (P_{1,2} \lor P_{1,1}) = true \land (false \lor true) = true \land true = true$

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 $[i,j]$.

$\neg P_{1,1}$
$\neg B_{1,1}$
$B_{2,1}$

“A square is breezy if and only if there is an adjacent pit”
Truth tables for inference

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_5$</th>
<th>$P_6$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>$R_6$</th>
<th>$R_7$</th>
<th>$R_8$</th>
<th>$KB$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</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>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</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>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</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>
<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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</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>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</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>
<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>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Enumerate rows (different assignments to symbols), if $KB$ is true in row, check that $\alpha$ is too.

Inference by enumeration

Depth-first enumeration of all models is sound and complete.

```plaintext
function 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 $\rightarrow$ a list of the proposition symbols in KB and $\alpha$
return TT-CHECK-ALL(KB, $\alpha$, symbols, [])

function TT-CHECK-ALL(KB, $\alpha$, symbols, model) returns true or false
if EMPTY?(symbols) then
    if PL-TRUE?(KB, model) then return PL-TRUE?(\alpha, model)
    else return true
else do
    $P \rightarrow$ FIRST(symbols); rest $\rightarrow$ REST(symbols)
    return TT-CHECK-ALL(KB, $\alpha$, rest, EXTEND(P, true, model)) and
    TT-CHECK-ALL(KB, $\alpha$, rest, EXTEND(P, false, model))
```

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

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
i.e., prove $\alpha$ by reductio ad absurdum

Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules
- Legitimate (sound) generation of new sentences from old
- Proof $\rightarrow$ 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., min-conflicts-like hill-climbing algorithms

Logical equivalence

Two sentences are logically equivalent iff true in the 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 $\lor$
| $(\alpha \land \beta) \land \gamma$ | $(\beta \land \alpha) \land \gamma$ | associativity of $\land$
| $(\alpha \lor \beta) \lor \gamma$ | $(\beta \lor \alpha) \lor \gamma$ | associativity of $\lor$
| $\neg(\neg \alpha)$ | $\alpha$ | double-negation elimination
| $\alpha \Rightarrow \beta$ | $(\neg \alpha) \Rightarrow \neg \beta$ | contraposition
| $(\alpha \Rightarrow \beta)$ | $\neg \alpha \lor \beta$ | implication elimination
| $(\alpha \Rightarrow \beta)$ | $(\neg \beta \lor \alpha)$ | biconditional elimination
| $\neg(\neg \alpha \land \beta)$ | $\neg \neg \alpha \land \neg \beta$ | De Morgan
| $\neg(\neg \alpha \lor \beta)$ | $\neg \neg \alpha \lor \neg \beta$ | De Morgan
| $(\alpha \land (\beta \lor \gamma))$ | $((\alpha \land \beta) \lor (\alpha \land \gamma))$ | 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 $\rightarrow$
- $\iff$ proposition symbol; or
- $\iff$ (conjunction of symbols) $\Rightarrow$ symbol
  - e.g., $C \land (B \Rightarrow A) \land (C \land D) \Rightarrow B$

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.
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 $PLFCEntails(KB, q)$ returns true or false
inputs: $KB$, the knowledge base, a set of propositional Horn clauses
$q$, the query, a proposition symbol

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 in $KB$

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 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 $a_1 \land \ldots \land a_k \implies b$ 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 $m$

   General idea: construct any model of $KB$ by sound inference, check $a$

Backward chaining

Idea: 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

Backward chaining example

Backward chaining example

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)

\begin{align*}
\text{set of clauses} & : \{ \ell_1 \lor \cdots \lor \ell_k, \ m_1 \lor \cdots \lor m_m \} \\
\text{where } \ell_i \text{ and } m_j \text{ are complementary literals. E.g.,} & : \{ P \lor \neg \phi \land \neg \psi, \ \neg P \lor \psi \}
\end{align*}

Resolution inference rule (for CNF): complete for propositional logic

\begin{align*}
\ell_1 \lor \cdots \lor \ell_k, \ & m_1 \lor \cdots \lor m_m \\
\text{new clauses } & : \{ \ell \lor \neg \delta \land \neg \epsilon \land \delta \lor \epsilon \}
\end{align*}

Resolution is sound and complete for propositional logic

Conversion to CNF

- **Resolution example**
  \[ KB = (B_{1,1} \leftrightarrow (P_{1,2} \lor P_{2,1}) \land \neg B_{1,1} \land \neg P_{1,2}) \]

- **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

  Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

  Forward, backward chaining are linear-time, complete for Horn clauses

  Resolution is complete for propositional logic

  Propositional logic lacks expressive power

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

  \begin{verbatim}
  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
  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
  return false
  \end{verbatim}