Logical agents

Chapter 7
Outline

- Knowledge-based agents
- Logic in general—models and entailment
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution
Knowledge bases

Inference engine

Knowledge base

\[ \text{Knowledge base} = \text{set of sentences in a formal language} \]

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
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

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
Knowledge Representation and Reasoning

◊ Represent knowledge in some formal language – Propositional logic
  – First-order logic
  – Probabilistic/Statistical models
  – Probabilistic First-order logic

◊ Think of represented knowledge as complete knowledge about the domain

◊ **Inference**: Answer a query posed over the model (deduction)

◊ **Learning**: Induce a model in a particular formal language that best fits the data (and possibly some prior assumptions)
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 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)
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 *α* if *α* is true in *m*.

*M(α)* is the set of all models of *α*.

Then *KB |= α* if and only if *M(KB) ⊆ M(α)*.

E.g. *KB = Giants won and Reds won*.

*α = Giants won*.
**Inference**

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

Consequences of \( KB \) are a haystack; \( \alpha \) is a needle.
Entailment = needle in haystack; inference = finding it

**Soundness:** \( i \) is sound if

\[ \text{whenever } KB \vdash_i \alpha, \text{ it is also true that } KB \models \alpha \]

**Completeness:** \( i \) is complete if

\[ \text{whenever } KB \models \alpha, \text{ it is also true that } KB \vdash_i \alpha \]

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, $S_1 \wedge S_2$ is a sentence (conjunction)

If $S_1$ and $S_2$ are sentences, $S_1 \vee S_2$ 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)
Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. \( P_{1,2} \), \( P_{2,2} \), \( P_{3,1} \)  
\( true \), \( true \), \( false \)

(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 \textbf{and} \( S_2 \) is true
- \( S_1 \lor S_2 \) is true iff \( S_1 \) is true \textbf{or} \( S_2 \) is true
- \( S_1 \Rightarrow S_2 \) is true iff \( S_1 \) is false \textbf{or} \( S_2 \) is true
  i.e., is false iff \( S_1 \) is true \textbf{and} \( S_2 \) is false
- \( S_1 \Leftrightarrow S_2 \) is true iff \( S_1 \Rightarrow S_2 \) is true \textbf{and} \( S_2 \Rightarrow S_1 \) is true

Simple recursive process evaluates an arbitrary sentence, e.g.,
\( \neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) = true \land (false \lor true) = true \land true = true \)
### 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>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>true</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>
Inference by enumeration

Depth-first enumeration of all models is sound and complete

**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

$symbols \leftarrow$ 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**

$P \leftarrow$ First($symbols$); $rest \leftarrow$ 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
Logical equivalence

Two sentences are logical equivalent iff true in same models:
\[ \alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha \]

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg\alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \leftrightarrow \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg\alpha \lor \neg\beta) \quad \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg\alpha \land \neg\beta) \quad \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
Validity and satisfiability

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

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
– \textbf{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 \textit{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
Forward and backward chaining

Horn Form (restricted)

\[ KB = \text{conjunction of Horn clauses} \]

Horn clause =

\begin{itemize}
  \item \text{proposition symbol; or}
  \item \text{(conjunction of symbols) } \Rightarrow \text{ symbol}
\end{itemize}

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

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

\[
\frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta}{\beta}
\]

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

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
function PL-FC-Entails?\((KB, q)\) 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
    \(p \leftarrow \text{POP}(agenda)\)
    unless \(inferred[p]\) do
        \(inferred[p] \leftarrow true\)
    for each Horn clause \(c\) in whose premise \(p\) appears do
        decrement \(count[c]\)
        if \(count[c] = 0\) then do
            if \(\text{HEAD}[c] = q\) then return true
            \(\text{Push(HEAD}[c], agenda)\)
    return false
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[ \begin{align*}
    P & \Rightarrow Q \\
    L \land M & \Rightarrow P \\
    B \land L & \Rightarrow M \\
    A \land P & \Rightarrow L \\
    A \land B & \Rightarrow L \\
    A & \\
    B &
\end{align*} \]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[ P \implies Q \]
\[ L \land M \implies P \]
\[ B \land L \implies M \]
\[ A \land P \implies L \]
\[ A \land B \implies L \]
\[ A \]
\[ B \]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

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

**General idea**: construct any model of $KB$ by sound inference, check $\alpha$
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

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

Equations:

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[
P \Rightarrow Q
\]
\[
L \land M \Rightarrow P
\]
\[
B \land L \Rightarrow M
\]
\[
A \land P \Rightarrow L
\]
\[
A \land B \Rightarrow L
\]
\[
A
\]
\[
B
\]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

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

clauses

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

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

\[
\ell_1 \lor \cdots \lor \ell_i, m_1 \lor \cdots \lor m_j
\]

\[
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\]

where \(\ell_i\) and \(m_j\) are complementary literals.

Resolution is sound and complete for propositional logic
Conversion to CNF

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

1. Eliminate \( \iff \), replacing \( \alpha \iff \beta \) with \( (\alpha \implies \beta) \land (\beta \implies \alpha) \).

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

2. Eliminate \( \implies \), replacing \( \alpha \implies \beta \) 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 (\( \lor \) over \( \land \)) and flatten:

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

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

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 $\leftarrow$ the set of clauses in the CNF representation of $KB \land \neg \alpha$
new $\leftarrow \{ \}$

loop do

for each $C_i, C_j$ in clauses do

resolvents $\leftarrow$ PL-Resolve($C_i, C_j$)

if resolvents contains the empty clause then return true
new $\leftarrow$ new $\cup$ resolvents

if new $\subseteq$ clauses then return false

clauses $\leftarrow$ clauses $\cup$ new
Resolution example

\[ KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \alpha = \neg P_{1,2} \]
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

Forward, backward chaining are linear-time, complete for Horn clauses. Resolution is complete for propositional logic.

Propositional logic lacks expressive power.