Logical agents
Outline

♦ Wumpus world (intro. to reasoning with an example game requiring reasoning)

♦ Knowledge base agents

♦ Representation, Reasoning, and Logic

♦ Propositional (Boolean) logic

♦ Logic in general—models and entailment

♦ Equivalence, validity, satisfiability

♦ Inference rules and theorem proving
  – forward chaining
  – backward chaining
  – resolution
Percepts Breeze, Glitter, Smell

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

Goals Get gold back to start without entering pit or wumpus square

Environment
- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter if and only if gold is in the same square
- Shooting kills the wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up the gold if in the same square
- Releasing drops the gold in the same square
Wumpus world characterization - Review

Is the world deterministic??

Is the world fully accessible??

Is the world static??

Is the world discrete??
Wumpus world characterization - Review

Is the world deterministic?? Yes—outcomes exactly specified
Is the world fully accessible?? No—only local perception
Is the world static?? Yes—Wumpus and Pits do not move
Is the world discrete?? Yes
After percept [None, None, None, None, None, None]
[Stench Brz. Glitter Bump Scream]
Uncertainty

There may be situations where there are no guaranteed safe moves. In that case, we will consider the likelihoods of events (later).

But for now, we assume that we are dealing with problems that can be solved with logical thinking and there is no chance involved (i.e. no remaining uncertainty).
Breeze in (1,2) and (2,1) ⇒ no safe actions

Smell in (1,1) ⇒ cannot move
How can we get an agent do what we did? That is, how can it know whether it is OK to move to [2,1]?

Notice that an exhaustive search is not an option here because the agent doesn't want to die.

The agent needs a formal inference mechanism to derive valid conclusions even when it does not know what the interpretation is (the computer does not need to know the meaning of OK, or Wumpus or Pit... )
We will use a formal logic to extend the capacity of our agents by endowing them with the capacity for general logical reasoning.

Formal logic: formal languages for representing information such that conclusions can be drawn.
Knowledge based agents

A knowledge based agent needs to know the current state of the world, how to infer unseen properties of the world, how the world evolves over time, what it wants to achieve, and what its own actions do.

Specifically, the agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Deduce hidden properties of the world
- Update internal representations of the world
- Deduce appropriate actions
Knowledge bases - 6.1

Knowledge base is a set of representations of facts (sentences) about the world, expressed in knowledge representation language.**

<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>

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
Knowledge bases

Agents can be viewed at the **knowledge level**
i.e., what they know, regardless of how implemented

The **implementation level** may represent the knowledge in various forms:
e.g. UcakSeferi(Istanbul,Samsun) can be represented as a string in a
list of strings or as 1 in a 2D table indexed by location pairs etc.

But we don’t care about the implementation details of the KB.
We are interested in new sentences that we can infer from the knowledgebase and the new percepts. E.g. seeing that there is no breeze in [1,1], can we infer that there is no wumpus in [1,2]? We will use the word entailment, which means that one thing follows from another.

Knowledge base $KB$ entails sentence $\alpha$ ($KB \models \alpha$) if and only if $\alpha$ is true in all worlds where $KB$ is true.

E.g., the KB containing “GS won” and “XY lost” entails ”GS won”

E.g., $x + y = 4$ entails $4 = x + y$

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

We say that $m$ 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. $KB = \text{GS won and XY lost}$
\[ \alpha = \text{GS won} \]

Note: Difference between *world* and *model*

"model" is used interchangeably with "world" in the general sense;
but a "model" of a particular sentence is a "world" in which that sentence
is True)
Entailment in the wumpus world

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

Consider all possible assignments to ?s assuming only pits

3 Boolean choices  ⇒  8 possible worlds
$KB = \text{wumpus-world rules} + \text{observations}$
\[ KB = \text{wumpus-world rules + observations} \]

\[ \alpha_1 = \text{“[1,2] is safe”, } KB \models \alpha_1, \text{ proved by model checking} \]
$K B = \text{wumpus-world rules} + \text{observations}$
$KB = \text{wumpus-world rules + observations}$

$\alpha_2 = \text{“[2,2] is safe”, } KB \not\models \alpha_2$
We know of some knowledge representation languages:

- Programming languages

In a programming language you can say that "there is a pit in [2,2]" by `world[2,2] = pit`, but you cannot say "either there is a pit in [2,1] or in [3,2]" or that "there is a wumpus in some square"

- Propositional logic

- First-order logic

- Natural languages are very expressive, but also too flexible/ambiguous

We will concentrate on first-order logic: basis of most representation schemes in AI. It is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure (complete: the procedure will answer any question whose answer follows from what is known by the $KB$.)
Logic

Logic is a formal language for representing information such that conclusions can be drawn.

Syntax defines the elements and possible sentences in the language.

Semantics define the “meaning” of sentences;

Consider the language of arithmetic:

Syntax:

\[ x + 2 \geq y \] is a sentence; \[ x + y + \] is not a sentence

Semantics:

\[ x + 2 \geq y \] 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 \( x = 0, \ y = 6 \)
Propositional logic is the simplest logic:

A sentence can be one of the following:

◊ Constant propositions True or False

◊ Propositions such as P or Q which may have value True or False

◊ Sentences formed by simpler ones using logical connectives:

If $S$ is a sentence, $\neg S$ is a sentence

If $S_1$ and $S_2$ is a sentence, $S_1 \land S_2$ is a sentence

If $S_1$ and $S_2$ is a sentence, $S_1 \lor S_2$ is a sentence

If $S_1$ and $S_2$ is a sentence, $S_1 \Rightarrow S_2$ is a sentence

If $S_1$ and $S_2$ is a sentence, $S_1 \Leftrightarrow S_2$ is a sentence
Propositional logic: Semantics

Assume you are given a model, $m$, specifying the truth settings (true/false) for each proposition symbol:

E.g. $A$, $B$, $C$

$\begin{array}{ccc}
\text{True} & \text{True} & \text{False}
\end{array}$

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
- $S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true
(P implies Q) is the only one a bit unintuitive: if P is false, the implication is basically irrelevant. So, simply look to see if Q is true whenever P is.
Logical equivalence

Two sentences are **logically equivalent** iff true in same models:

\[ \alpha \equiv \beta \text{ if and only if } \alpha \models \beta \text{ and } \beta \models \alpha \]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\alpha \land \beta) \equiv (\beta \land \alpha))</td>
<td>commutativity of (\land)</td>
</tr>
<tr>
<td>((\alpha \lor \beta) \equiv (\beta \lor \alpha))</td>
<td>commutativity of (\lor)</td>
</tr>
<tr>
<td>((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma)))</td>
<td>associativity of (\land)</td>
</tr>
<tr>
<td>((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma)))</td>
<td>associativity of (\lor)</td>
</tr>
<tr>
<td>((\alpha \equiv \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)))</td>
<td>biconditional elimination</td>
</tr>
<tr>
<td>((\alpha \lor \beta \lor \alpha) \equiv (\neg \alpha \lor \neg \beta))</td>
<td>De Morgan</td>
</tr>
<tr>
<td>((\alpha \lor \beta \lor \beta) \equiv (\neg \alpha \land \neg \beta))</td>
<td>De Morgan</td>
</tr>
<tr>
<td>((\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)))</td>
<td>distributivity of (\land) over (\lor)</td>
</tr>
<tr>
<td>((\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)))</td>
<td>distributivity of (\lor) over (\land)</td>
</tr>
</tbody>
</table>
Validity and Satisfiability

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

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., $A \land \neg A$

Validity is connected to inference:
$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

Satisfiability is connected to inference:
$KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable
i.e., prove $\alpha$ by reductio ad absurdum - we will see this later
Validity and Inference

Truth tables can be used to test for valid sentences.

E.g. Show that \((P \lor H) \land \neg H \Rightarrow P\) is a valid sentence. How?
Validity and Inference

Show that \(((P \lor H) \land \neg H) \Rightarrow P\) is a valid sentence.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(P)</td>
<td>(H)</td>
<td>(P \lor H)</td>
<td>((P \lor H) \land \neg H)</td>
<td>(((P \lor H) \land \neg H) \Rightarrow P)</td>
</tr>
<tr>
<td>False</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>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

If the sentence is True in all possible worlds (i.e., rows indicating the truth assignments to all the variables), it is Valid.
Proof/Inference methods

Assume that we first knew that "there is a Pit in 2x2 or 2x1" (from the previous slide: $P \lor H$), and then we learned "there is no Pit in 2x1" ($\neg H$).

Then we could conclude that "there is a Pit in 2x2" ($P$).

Proof = a sequence of inference rule applications that shows a conclusion (e.g. pit in 2x2).
There are roughly two kinds of proof methods:

**Model checking:**
- Truth table enumeration (sound and complete for propositional logic)
- Heuristic search in model space (sound but incomplete)

**Application of inference rules**
- Legitimate (sound) generation of new sentences from old
- Can use inference rules as operators in a standard search alg.
Propositional inference: Enumeration method

Let $\alpha = A \lor B$ and $KB = (A \lor C) \land (B \lor \neg C)$

Is it the case that $KB \models \alpha$?
Check all possible models—$\alpha$ must be true wherever $KB$ is true

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$A \lor C$</th>
<th>$B \lor \neg C$</th>
<th>$KB$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Propositional inference: Enumeration method

Let $\alpha = A \lor B$ and $KB = (A \lor C) \land (B \lor \neg C)$

Is it the case that $KB \models \alpha$?
Check all possible models: $\alpha$ must be true wherever $KB$ is true

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$A \lor C$</th>
<th>$B \lor \neg C$</th>
<th>$KB$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</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>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</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>True</td>
<td>True</td>
<td>True</td>
<td>True</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>

Note that this is equivalent to proving that the sentence

$((A \lor C) \land (B \lor \neg C) \Rightarrow A \lor B$ is a valid sentence!
Complexity of inference

Truth table method: $2^N$ rows of table for any proof involving $N$ symbols

Linear space requirements though using Depth-First search.
PL Inference: Enumeration method

Sorry for the watermark. You should study the same figure in Figure 7.10.
Inference by enumeration

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

```
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-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true.
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
        TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))

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

An inference procedure that generates only entailed sentences is called **sound**.

An inference procedure that generates all sentences that can be entailed is called **complete**.

Formally:

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

**Soundness**: \( i \) is sound if whenever \( KB \vdash_i \alpha \), it is also true that \( KB \models \alpha \).

**Completeness**: \( i \) is complete if whenever \( KB \models \alpha \), it is also true that \( KB \vdash_i \alpha \).
Rules of Inference

Instead of enumeration method, we can also use inference rules that we know are sound. We list these rules as:

\[
\frac{\alpha}{\beta}
\]

**Notation:** Whenever a sentence in the KB matches the pattern above the line (\(\alpha\)), the inference rule can conclude the sentence below the line (\(\beta\)).
Inference rules for propositional logic

Modus Ponens (or implication elimination)

\[
\frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \land \cdots \land \alpha_n}{\beta}
\]
Inference rules for propositional logic

Modus Ponens (or implication elimination)

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

And-elimination (from a conjunction, you can infer any of the conjuncts)

\[
\frac{\alpha_1 \land \cdots \land \alpha_n}{\alpha_i}
\]
Inference rules for propositional logic

**Modus Ponens** (or implication elimination)

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

**And-elimination** (from a conjunction, you can infer any of the conjuncts)

\[
\frac{\alpha_1 \land \cdots \land \alpha_n}{\alpha_i}
\]
Inference rules for propositional logic

Modus Ponens (or implication elimination)

\[ \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta \]
\[ \beta \]

And-elimination (from a conjunction, you can infer any of the conjuncts)

\[ \alpha_1 \land \cdots \land \alpha_n \]
\[ \alpha_i \]

Unit resolution (from a disjunction, if one of the disjuncts are false, you can infer the other one is true)

\[ \alpha \lor \beta \quad \neg \beta \]
\[ \alpha \]
Assume the following situation:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>G</th>
<th>P</th>
<th>S</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>OK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td>P!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td>OK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A = Agent  
B = Breeze  
G = Glitter, Gold  
OK = Safe square  
P = Pit  
S = Stench  
V = Visited  
W = Wumpus

How does the agent know where to go?
Entailment Using Inference Rules

Go over the example in AIMA 2nd ed. pg 212 (or the example in AIMA 1st ed pp. 175-176).

Start with KB and see what other sentences you can entail. If you reach the sentence you want to obtain (e.g. \( \neg P_{12} \land \neg P_{21} \)), you are done.
Restricted Forms: Horn Form

**Horn Form (restricted):** A KB is said to be in Horn Form if it is a conjunction of Horn clauses.

Horn clause =
- ♦ proposition symbol; or
- ♦ (conjunction of symbols) \( \Rightarrow \) symbol

**Alternative definition:** a Horn clause is a disjunction of literals of which at most one is positive

Example KB: \( C \land (B \Rightarrow A) \land (C \land D \land E \Rightarrow B) \)

Definition: Body (premise) and Head (conclusion).
Restricted Forms: Horn Form

The Horn form seems very restricted, but many real databases can be written in Horn form. How?

◊ Definite clauses: with exactly one positive literal

e.g.: \((C \land D \Rightarrow B)\)

◊ Facts: a definite clause with no negative literals simply asserts a given proposition

e.g.: \(C\)

◊ Integrity constraints: a Horn clause with no positive literals

e.g.: \(\neg(W11 \land W22)\)
Restricted Forms: Horn Form

How can you convert each sentence into an implication?

\[(W_{11} \land W_{22}) \Rightarrow False\]
Restricted Forms: Horn Form

Modus Ponens is complete for Horn KBs (not in general)

\[
\begin{align*}
\alpha_1, \ldots, \alpha_n, & \quad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta \\
\beta & \quad \frac{\alpha_1, \ldots, \alpha_n,}{\alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta}
\end{align*}
\]

Can be used with forward chaining or backward chaining. These algorithms are very natural and run in linear time in the size of the KB.

They are also sound (easy to show) and complete (slightly more complicated argument).
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

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
    \(p \leftarrow \text{POP}(\text{agenda})\)
    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 HEAD[\(c\)] = \(q\) then return true
                Push(HEAD[\(c\)], agenda)
        end for
    end unless
return false
Forward chaining example
Forward chaining example

\[\text{Diagram with nodes labeled A, B, L, M, P, Q and numbered edges connecting them.}\]
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example

A B
L M P Q
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

The algorithm is essentially identical to the AND-OR-GRAPH search given in Fig. 4.11.
Backward chaining example

Diagram showing a backward chaining example with nodes labeled Q, P, M, L, A, and B.
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

A → Q → P → M → L → A → B

AIMA 2nd ed - Chapter 7  75
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
Complete Set of Inference rules for propositional logic

**Modus Ponens** (or implication elimination)

\[
\alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta \\
\beta
\]

**And-elimination** (from a conjunction, you can infer any of the conjuncts)

\[
\alpha_1 \land \cdots \land \alpha_n \\
\alpha_i
\]

**Unit resolution** (from a disjunction, if one of the disjuncts are false, you can infer the other one is true)

\[
\alpha \lor \beta, \quad \neg \beta \\
\alpha
\]

**Resolution** (for CNF): complete for propositional logic

\[
\alpha \lor \beta, \quad \neg \beta \lor \gamma \\
\alpha \lor \gamma
\]
Remember: Validity and Satisfiability

A sentence is valid (is a tautology) if it is true in all models

  e.g., \( A \lor \neg A \), \( A \Rightarrow A \), \( (A \land (A \Rightarrow B)) \Rightarrow B \)

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., \( A \land \neg A \)

Validity is connected to inference:

\[ KB \models \alpha \text{ if and only if } (KB \Rightarrow \alpha) \text{ is valid} \]

Satisfiability is connected to inference:

\[ KB \models \alpha \text{ if and only if } (KB \land \neg \alpha) \text{ is unsatisfiable} \]

i.e., prove \( \alpha \) by *reductio ad absurdum* - we will see this later
Normal forms

Other approaches to inference use syntactic operations on sentences which are often expressed in standardized forms:

**Horn Form (restricted)**

*conjunction of Horn clauses* (clauses with \( \leq 1 \) positive literal)

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

Often written as set of implications:

\( B \Rightarrow A \) and \((C \land D) \Rightarrow B\)

**Disjunctive Normal Form (DNF—universal)**

*disjunction of conjunctions of literals*

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

**Conjunctive Normal Form (CNF—universal)**

*conjunction of disjunctions of literals*

E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)
Resolution requires the Conjunctive Normal Form.

Every sentence can be written in CNF.
Resolution

Conjunctive Normal Form (CNF—universal)

\(\text{conjunction of disjunctions of literals}\)

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

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

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

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

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

\[
\begin{array}{c}
P_{1,3} \lor P_{2,2}, \\
\neg P_{2,2}
\end{array}
\]

\[
P_{1,3}
\]

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 \( (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \).

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

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 (\( \lor \) over \( \land \)) and flatten:

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

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

function PL-Resolve($KB$, $\alpha$) 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
Resolution example

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

Add the negation of the sentence you want to prove (e.g. \( \neg \neg P_{1,2} \)) and see if you reach a contradiction:
Resolution example

\[ KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \land \neg P_{1,2} \]
Satisfiability Problem

Satisfiability problem (is this sentence satisfiable?) is at the core of many general problems (e.g. constraint satisfaction problems).

Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.

However, determining the satisfiability of sentences in propositional logic was the first problem proved to be NP-complete (in the worst case, it doesn’t have a polynomial time algorithm).
Local search algorithms

We can use local search algorithms for satisfiability problems, as well.

Since each clause in the sentence needs to be satisfied, evaluation function should count the number of unsatisfied clauses (e.g. min-conflict heuristic).

WALKSAT algorithm is one of the most effective algorithms.
WALKSAT algorithm

- In each iteration, pick an unsatisfied clause and picks a symbol to flip.
- Two ways to decide how to choose the symbol:
  - randomly select the symbol from the clause and flip it with probability p
  - pick one that minimizes the number of unsatisfied clauses in the new state
Local search algorithms

In underconstrained situations, solutions are easy to find.

Ex. A 3-CNF with 5 symbols and 5 clauses:

\[
(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C') \land (\neg C' \lor \neg B \lor E) \\
\land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C')
\]

How many solutions?

What makes it a hard satisfiability problem?

What if WALKSAT returns failure?
Local search algorithms

In underconstrained situations, solutions are easy to find.

Ex. A 3-CNF with 5 symbols and 5 clauses:

\[
(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C') \land (\neg C' \lor \neg B \lor E) \\
\land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C')
\]

How many solutions?

16 of the 32 possible worlds are models for this sentence!

What makes it a hard satisfiability problem?

How constrained it is (w.r.t number of constraints and number of variables)

What if WALKSAT returns failure?

We cannot tell if the sentence is unsatisfiable or just WALKSAT needs more time.
Shortcomings of PL

Lack of Expressive Efficiency:

We need to state the general rule about breeze and pits as a separate sentence for each square:

\[ B_{11} \Rightarrow P_{21} \lor P_{12} \]

\[ B_{21} \Rightarrow P_{11} \lor P_{22} \lor P_{31} \]
Shortcomings of PL - Fluents

**Fluents:** changing aspects of the world

E.g. Keeping track of location and orientation.

Let $L_{ij}$ be a location proposition. Then we would like to say something like:

$L_{11} \land \text{FacingRight} \land \text{Forward} \Rightarrow L_{21}$

But then we will have both $L_{11}$ and $L_{21}$ in the database!
**Solution:** Associate propositions with time steps:

\[ L_{11}^0, FacingEast^0, HaveArrow^0 \ldots \]

After associating time with prepositions, we need axioms to keep track of these fluents:

\[ L_{xy}^t \Rightarrow (\text{Breeze}^t \iff \text{Breeze}_{xy}) \]

**Effect axioms:**

\[ L_{11}^0 \land FacingEast^0 \land Forward^0 \Rightarrow (L_{21}^1 \land \neg L_{11}^1) \]

We would need one such sentence for each possible time step and each possible square and orientation!
Shortcomings of PL - Frame Problem

We still have other problems. The effect axioms fail to state what is unchanged as the result of an action.

E.g. That the agent still has an arrow.

One possible solution is to add frame axioms explicitly asserting all the propositions that remain the same:

\[\text{Forward}^t \Rightarrow (\text{HaveArrow}^t \iff \text{HaveArrow}^{t+1})\]

\[\text{Forward}^t \Rightarrow (\text{WumpusAlive}^t \iff \text{WumpusAlive}^{t+1})\]

...
The proliferation of frame axioms is very inefficient and this problem was a significant problem for AI researchers.

**Solution:** Focusing from writing axioms about actions to writing axioms about fluents (*successor-state axioms*): $\text{HaveArrow}^{t+1} \iff (\text{HaveArrow}^{t} \land \neg \text{Shoot}^{t})$

\[
L_{11}^{t+1} \iff (L_{11}^{t} \land (\neg \text{Forward}^{t} \lor \text{Bump}^{t+1}))
\lor (L_{12}^{t} \land (\text{FacingSouth}^{t} \land \text{Forward}^{t}))
\lor (L_{21}^{t} \land (\text{FacingWest}^{t} \land \text{Forward}^{t})).
\]
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. Propositional logic suffices for some of these tasks.

Truth table method is sound and complete for propositional logic.

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

Propositional logic lacks expressive power