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
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
Wumpus World PEAS description

**Environment:**
One wumpus, one heap of gold

\[ P(\text{pit}) = 0.2 \]
for each square

Squares next to wumpus are smelly

Shooting into wumpus’s square kills it

Shooting uses up the only arrow

Squares next to pit are breezy

Glitter iff the gold is in your square

Grabbing picks it up

Releasing drops it

**Performance measure:**
gold +1000, death −1000, −1 per step, −10 for using the arrow

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

**Sensors:** Breeze, Glitter, Smell
Wumpus world characterization

Fully observable?
Wumpus world characterization

*Fully observable?* No—only *local* perception

*Deterministic?*
Wumpus world characterization

**Fully observable?**  No—only local perception

**Deterministic?**  Yes—outcomes exactly specified

**Episodic?**
Wumpus world characterization

**Fully observable?** No—only local perception

**Deterministic?** Yes—outcomes exactly specified

**Episodic?** No—sequential at the level of actions

**Static?**
Wumpus world characterization

**Fully observable?**  No—only local perception

**Deterministic?**  Yes—outcomes exactly specified

**Episodic?**  No—sequential at the level of actions

**Static?**  Yes—Wumpus, pits, and gold do not move

**Discrete?**
Wumpus world characterization

Fully observable? No—only local perception

Deterministic? Yes—outcomes exactly specified

Episodic? No—sequential at the level of actions

Static? Yes—Wumpus, pits, and gold do not move

Discrete? Yes

Single-agent?
Wumpus world characterization

**Fully observable?** No—only local perception

**Deterministic?** Yes—outcomes exactly specified

**Episodic?** No—sequential at the level of actions

**Static?** Yes—Wumpus, pits, and gold do not move

**Discrete?** Yes

**Single-agent?** Yes—Wumpus is essentially a natural feature
Exploring a wumpus world

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td>A</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CMSC 421: Chapter 7  13
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Other tight spots

Breeze in (1,2) and (2,1)
⇒ no safe actions

\[ P(\text{pit in (2,2)}) \approx 0.86 \]
\[ P(\text{pits in (1,3) and (3,1)}) \approx 0.31 \]

In a later chapter we’ll see how to compute this

Smell in (1,1)
⇒ cannot move safely
Can use a strategy of coercion:
shoot straight ahead
wumpus was there ⇒ dead ⇒ safe
wumpus wasn’t there ⇒ safe
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 at least as big as 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., if a KB contains “Maryland won” and “Duke won”,
the KB entails “Maryland won or* Duke 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)

*The “or” is inclusive, not exclusive.
Models

A *model* is a formally structured world in 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 = Maryland won and Duke won*  
*α = Maryland won*
Entailment in the wumpus world

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

For now, ignore the wumpus and gold. Which of the ?s are pits?

For each possible combination of pit locations, check whether it’s a model.

3 Boolean choices  \( \Rightarrow \)  8 possible models
Wumpus models

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

Eight possible combinations of pit locations: which ones are models of $KB$?
$KB = \text{wumpus-world rules} + \text{observations}$

Three models
Wumpus models

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

\[ \alpha_1 = "[1,2] \text{ is safe}, \quad KB \models \alpha_1, \text{ proved by model checking} \]
Wumpus models

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

$\alpha_2 = \text{“}[2,2] \text{ is safe”}, \ KB \not\models \alpha_2$
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
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 $KB \vdash_i \alpha$

Model checking (what we just did) is one kind of inference procedure,
but not the only one

Model checking is sound
Model checking is complete if the set of all possible models is finite
Preview of where we’re going

Later, we’ll 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 $KB$.

But first, let’s look at propositional logic.
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)
Propositional logic: Semantics

Each model specifies a true/false value for every proposition symbol

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

\[
\begin{array}{c c c}
\text{true} & \text{true} & \text{false} \\
\end{array}
\]

(8 possible models, can be enumerated automatically)

Rules for evaluating truth with respect to a model \( m \):

- \( \lnot S \) 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

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

\[
\lnot P_{1,2} \land (P_{2,2} \lor P_{3,1}) = \text{true} \land (\text{false} \lor \text{true}) = \text{true} \land \text{true} = \text{true}
\]
## Truth tables for connectives

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

$B_{1,1} \iff (P_{1,2} \lor P_{2,1})$
$B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})$

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

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$KB$</th>
<th>$\alpha_1$</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>
<td>don't care</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>false</td>
<td>don't care</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>
<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>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>don't care</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>don't care</td>
</tr>
</tbody>
</table>

Model checking in propositional logic = inference using truth tables

Each row is a potential model: an assignment of truth values to symbols

if $KB$ is true in row, is $\alpha$ true too?
**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 ← 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 ← First(symbols); rest ← 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 *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>Equivalent Expression</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\alpha \land \beta)) \equiv (\beta \land \alpha)</td>
<td>commutativity of (\land)</td>
<td></td>
</tr>
<tr>
<td>((\alpha \lor \beta)) \equiv (\beta \lor \alpha)</td>
<td>commutativity of (\lor)</td>
<td></td>
</tr>
<tr>
<td>(((\alpha \land \beta) \land \gamma)) \equiv (\alpha \land (\beta \land \gamma))</td>
<td>associativity of (\land)</td>
<td></td>
</tr>
<tr>
<td>(((\alpha \lor \beta) \lor \gamma)) \equiv (\alpha \lor (\beta \lor \gamma))</td>
<td>associativity of (\lor)</td>
<td></td>
</tr>
<tr>
<td>(\neg(\neg\alpha)) \equiv \alpha</td>
<td>double-negation elimination</td>
<td></td>
</tr>
<tr>
<td>((\alpha \Rightarrow \beta)) \equiv (\neg\beta \Rightarrow \neg\alpha)</td>
<td>contraposition</td>
<td></td>
</tr>
<tr>
<td>((\alpha \Rightarrow \beta)) \equiv (\neg\alpha \lor \beta)</td>
<td>implication elimination</td>
<td></td>
</tr>
<tr>
<td>((\alpha \iff \beta)) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))</td>
<td>biconditional elimination</td>
<td></td>
</tr>
<tr>
<td>(\neg(\alpha \land \beta)) \equiv (\neg\alpha \lor \neg\beta)</td>
<td>De Morgan</td>
<td></td>
</tr>
<tr>
<td>(\neg(\alpha \lor \beta)) \equiv (\neg\alpha \land \neg\beta)</td>
<td>De Morgan</td>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>
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 at least one 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 = 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
Forward and backward chaining

**Horn Form**

The KB is a **conjunction** of Horn clauses.

A Horn clause is:

- a proposition symbol; or
- a conjunction of symbols **⇒** a symbol

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

This is a restricted subset of propositional logic.

E.g., the following is not a Horn clause, and can’t be translated into one:

\( A \lor B \)

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

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

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(c), number of c’s premises not yet inferred
                 inferred(c), whether or not c has been inferred
                 agenda, {all clauses that are ready to be inferred}

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

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

A
B

Diagram:

- P
- L
- M
- A
- B

Arrows indicate the direction of inference.
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

\[ P \implies Q \]
\[ L \land M \implies P \]
\[ B \land L \implies M \]
\[ A \land P \implies L \]
\[ A \land B \implies L \]
\[ A \]
\[ B \]
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
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. At $i$'th iteration of the “while” loop, can create a world $m_i$ as follows:
   - assign true to every atomic symbol that has been derived
   - assign false to all other atomic symbols.

2. There are only finitely many atomic sentences, so FC must reach a fixed point where no new atomic sentences are derived. Let $n$ be the iteration where this happens.

3. Every clause in the original $KB$ is true in $m_n$
   
   **Proof**: Suppose a clause $a_1 \land \ldots \land a_k \Rightarrow b$ is false in $m_i$.
   
   Then $a_1 \land \ldots \land a_k$ is true in $m_i$ and $b$ is false in $m_i$.
   
   Thus $\text{count}(b) = 0$, so $b$ will be inferred in a future iteration, so iteration $i$ isn’t a fixed point.
   
   Hence at the fixed point, $m_n$ is a model of $KB$.

4. If $q$ is atomic and $KB \models q$, then $q$ is true in every model of $KB$, including $m$. Hence FC must have derived $q$. 

Backward chaining

Idea: work backwards from the query $q$:
- to prove $q$ by BC,
  - check if $q$ is known already, or
  - recursively call BC to prove all premises of some rule concluding $q$

Avoid loops: check if new subgoal is already on the recursion 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

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

\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*}
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*

- data-driven algorithms can be used for 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): conjunction of disjunctions of literals

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

Resolution inference rule:

If \(\ell_i\) and \(m_j\) are negations of each other,

\[
\begin{align*}
\ell_1 \lor \cdots \lor \ell_i \lor \cdots \lor \ell_k, & \quad m_1 \lor \cdots \lor m_j \lor \cdots \lor m_n \\
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\end{align*}
\]

i.e., the disjunct of everything other than \(\ell_i\) and \(m_j\)

Example:

\[
\begin{align*}
P_{1,3} \lor P_{2,2}, & \quad \neg P_{2,2} \\
P_{1,3}
\end{align*}
\]
Resolution

Resolution is equivalent to Modus Ponens:

clauses from CNF: \( A \lor \neg B \quad B \lor \neg C \lor \neg D \)

rewrite as implications: \( \neg A \Rightarrow \neg B \quad \neg B \land C \Rightarrow \neg D \)

apply modus ponens:

\[\begin{align*}
\neg A \land C & \Rightarrow \neg D \\
\end{align*}\]

rewrite as clauses: \( A \lor \neg C \lor \neg D \)

Resolution is sound and complete for propositional logic

But to use it, you need to convert all your propositional statements to CNF
Conversion to CNF

There’s a breeze in (1,1) iff there’s a pit in (1,2) or (2,1):

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

1. Eliminate \( \iff \), by 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 \( \implies \), by 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:

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

Proof by contradiction: to prove $KB \Rightarrow \alpha$, show $KB \land \neg \alpha$ is unsatisfiable

```plaintext
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
    ;; compute all possible resolvents from clauses, and add them to clauses
    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
```

CMSC 421: Chapter 7   71
**Resolution example**

KB: there's a breeze in (1,1) iff there's a pit in (1,2) or (2,1);
and there's no breeze in (1,1)

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

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

\[
\alpha = \neg P_{1,2} \quad \text{we want to show there's no pit in (1,2)}
\]

\[
\neg \alpha = P_{1,2} \quad \text{suppose there is one (for proof by contradiction)}
\]
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.