Chapter 7

(Please turn your mobile devices to silent. Thanks!)
Last Time

- Constraint Satisfaction Problems (CSPs)
  - Can be viewed as DFS = backtracking search
    - Assign 1 value to 1 variable at every search tree level
- Specific heuristics to help
  - MRV / Degree / LCV
  - Forward Checking
  - Arc & Path & K Consistency
Knowledge-based agents

- Logic: models and entailment
- A simple logic: propositional (Boolean) logic
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution
Knowledge based agents

- Knowledge base (KB) = set of sentences in a formal language

- **Declarative** (as opposed to procedural) approach to build an agent:
  - 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 \text{KB-Agent}(\text{percept}) \text{ returns } \text{an action}

\text{static: } \text{KB}, \text{ a knowledge base}

\text{t, a counter, initially 0, indicating time}

\text{Tell}(\text{KB, Make-Percept-Sentence (percept, t)})

\text{action} \leftarrow \text{Ask}(\text{KB, Make-Action-Query(t)})

\text{Tell}(\text{KB, Make-Action-Sentence(action, t)})

\text{t} \leftarrow \text{t} + 1

\text{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

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

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

- **Sensors:** Stench, Breeze, Glitter, Bump, Scream
- **Actuators:** Turn Left, Turn Right, Forward, Grab, Release, Shoot
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Exploring a wumpus world
Logics

- **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 \( x = 0, y = 6 \)
Entropy

- Entailment means that one thing follows from another:
  \[ \text{KB} \models \alpha \]

- Knowledge base \( KB \) entails sentence \( \alpha \) if and only if \( \alpha \) is true in all worlds where \( KB \) is true
  - E.g., a KB containing “Today is sunny” and “Yesterday was rainy” entails “Either today is sunny or yesterday was rainy”
  - E.g., \( x+y = 4 \) entails \( 4 = x+y \)
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics
Logicians often think in terms of models ("possible worlds"), 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 $KB \models \alpha$ iff $M(KB) \subseteq M(\alpha)$.

E.g. $KB = \text{Today is sunny and yesterday was rainy}$

$\alpha = \text{Today is sunny}$
Entailment in the wumpus world

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

Consider possible models for $KB$ assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models
Wumpus world models
Wumpus models

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

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

- $KB = \text{wumpus-world rules} + \text{observations}$
- $\alpha_2 = "[2,2] \text{ is safe}"$, $KB \not\models \alpha_2$
Inference

- Define: $KB \models_i \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ by procedure } i$
  - **Soundness**: $i$ is sound if whenever $KB \models_i \alpha$, it is also true that $KB \models \alpha$
  - **Completeness**: $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \models_i \alpha$

- An inference procedure will answer any question whose answer follows from what is known by the $KB$.

“Entailment is like the needle (\(\alpha\)) being in the haystack (KB) and inference is like finding it”

We want to know: Is a set of inference operators complete and sound?
Completeness: $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \models_i \alpha$.

- An incomplete inference algorithm cannot reach all possible conclusions.
  - Equivalent to completeness in search (chapter 3).
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)
Each model specifies true/false for each proposition symbol

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

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

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

\[
\begin{align*}
\neg S & \text{ is true iff } S \text{ is false} \\
S_1 \land S_2 & \text{ is true iff } S_1 \text{ is true and } S_2 \text{ is true} \\
S_1 \lor S_2 & \text{ is true iff } S_1 \text{ is true or } S_2 \text{ is true} \\
S_1 \Rightarrow S_2 & \text{ is true iff } S_1 \text{ is false or } S_2 \text{ is true} \\
i.e., & \text{ is false iff } S_1 \text{ is true } S_2 \text{ is false} \\
S_1 \iff S_2 & \text{ is true iff } S_1 \Rightarrow S_2 \text{ is true and } S_2 \Rightarrow S_1 \text{ is true}
\end{align*}
\]

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

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

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</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>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</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 $[i, j]$.

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$

- How do we translate "Pits cause breezes in adjacent squares"?
**Truth tables for inference**

<table>
<thead>
<tr>
<th></th>
<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>1</td>
<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>true</td>
</tr>
<tr>
<td>2</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

$R_1 = \neg P_{1,1}$
$R_4 = \neg B_{1,1}$
$R_5 = B_{2,1}$

$\alpha_1 = \neg P_{1,2}$? (Is 1,2 safe from pits)?
Inference by enumeration

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

```plaintext
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))
```

- For $n$ symbols, time complexity is $O(2^n)$, space complexity is $O(n)$
Logical equivalence

- Two sentences are logically equivalent iff true in same models: \( \alpha \equiv \beta \) iff \( \alpha \models \beta \) 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
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 algorithm
    - Typically require transformation of sentences into a normal form
  - Model checking
    - truth table enumeration (always exponential in $n$)
    - improved backtracking, e.g., Davis-Putnam-Logemann-Loveland (DPLL)
    - heuristic search in model space (sound but incomplete)
      - e.g., min-conflicts like hill-climbing algorithms
Applying inference rules

Equivalent to a search problem

- KB state = node
- Inference rule application = edge

\[ \text{KB: } B, A \land D \land C, \ B \Rightarrow F \]

\[ \text{KB: } B, A \land D \land C, \ B \Rightarrow F, \ A \]

\[ \text{KB: } B, A \land D \land C, \ B \Rightarrow F, \ F \]
Resolution

Conjunctive Normal Form (CNF)
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):

\[
\begin{array}{c}
\ell_i \lor \ldots \lor \ell_k, \\
\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k, \\
\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_n
\end{array}
\]

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

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

Resolution is sound and complete for propositional logic
Soundness of Resolution

\[ \neg (l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i \]

\[ \neg m_j \Leftrightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n) \]

\[ \neg (l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n) \]

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

If \( l_i \) true, then \( m_j \) is false, hence \((m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\) must be true.

If \( m_j \) true, then \( l_i \) is false, hence \((l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k)\)
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 (\( \land \) over \( \lor \)) 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

```plaintext
function PL-Resolution(KB, \alpha) returns true or false
    clauses \leftarrow \text{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
    end loop
```

Resolution example

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$ (negate the premise for proof by refutation)

To think about: what does an empty proposition mean?
The power of false

- Given: \((P) \land (\neg P)\)
- Prove: \(Z\)

| \(\neg P\) | Given |
| \(P\)     | Given |
| \(\neg Z\) | Given |

Unsatisfiable

- Can we prove \(\neg Z\), using the givens above?
Forward and backward chaining

- **Horn Form** (restricted)
  - KB = conjunction of Horn clauses
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) ⇒ symbol
  - E.g., C ∧ (B ⇒ A) ∧ (C ∧ D ⇒ B)
- **Modus Ponens** (for Horn Form): complete for Horn KBs
  \[
  \alpha_1, \ldots, \alpha_n, \quad \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta
  \]

  \[
  \beta
  \]

- 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

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

```
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 ← 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 is sound and complete for Horn KB
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
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 $m$
   $$a_1 \land \ldots \land a_k \Rightarrow b$$
4. Hence $m$ is a model of $KB$
5. If $KB \models q$, $q$ is true in every model of $KB$, including $m$
Backward chaining

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
Backward chaining example
Backward chaining example
Backward chaining example

CS 3243 - Logical Agents
Backward chaining example
Backward chaining example
Backward chaining example
Forward vs. backward chaining

- FC is data-driven, 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
Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms
- DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - WalkSAT algorithm
The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination
   A clause is true if any literal is true.
   A sentence is false if any clause is false.

2. Pure symbol heuristic
   Pure symbol: always appears with the same "sign" in all clauses.
   e.g., In the three clauses (A ∨ ¬B), (¬B ∨ ¬C), (C ∨ A), A and B are pure, C is impure.
   Make a pure symbol literal true.

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.

What are correspondences between DPLL and in general CSPs?
The DPLL algorithm

function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns true or false

if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false
P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])

P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])

P ← FIRST(symbols); rest ← REST(symbols)
return DPLL(clauses, rest, [P = true|model]) or
DPLL(clauses, rest, [P = false|model])
The **WalkSAT** algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness
The **WalkSAT** algorithm

function WalkSAT(clauses, p, max-flips) returns a satisfying model or failure
inputs: clauses, a set of clauses in propositional logic
         p, the probability of choosing to do a “random walk” move
         max-flips, number of flips allowed before giving up

model ← a random assignment of true/false to the symbols in clauses
for i = 1 to max-flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    with probability p flip the value in model of a randomly selected symbol
    from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure

Let’s ask ourselves: Why is it **incomplete**?
Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,

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

\[ m = \text{number of clauses} \]
\[ n = \text{number of symbols} \]

- Hard problems seem to cluster near \( m/n = 4.3 \) (critical point)
Hard satisfiability problems
Hard satisfiability problems

- Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$
Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

\[ \neg P_{1,1} \]
\[ \neg W_{1,1} \]
\[ B_{x,y} \iff (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y}) \]
\[ S_{x,y} \iff (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y}) \]
\[ W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4} \]
\[ \neg W_{1,1} \lor \neg W_{1,2} \]
\[ \neg W_{1,1} \lor \neg W_{1,3} \]
\[ \ldots \]

\[ \Rightarrow 64 \text{ distinct proposition symbols, 155 sentences} \]
Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

- For every time $t$ and every location $[x,y]$,
  \[ L_{x,y} \land FacingRight_t \land Forward_t \implies L_{x+1,y} \]

- Rapid proliferation of clauses
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 w.r.t. 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

- The wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Resolution is complete for propositional logic
  Forward, backward chaining are linear-time, complete for Horn clauses
- Propositional logic lacks expressive power