Logical Agents

Chapter 7
Outline

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

• Due: Monday, May 23rd, in class.
• Chapter 6:
  – 6.1, 6.3, 6.4
• Chapter 7:
  – 7.3, 7.4, 7.6, 7.12
• Chapter 8:
  – 8.3, 8.6, 8.13
Knowledge bases

- 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 (procedures) 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

- **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:** Left turn, Right turn, Forward, Grab, Release, Shoot
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 and Pits do not move
- **Discrete** Yes
- **Single-agent?** Yes – Wumpus is essentially a natural feature
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
Exploring a wumpus world
Logic in general

- **Logics** are formal languages for representing information such that conclusions can be drawn.
- **Syntax** defines the sentences in the language.
- **Semantics** define the "meaning" of sentences;
  - i.e., define truth of a sentence in a world.

- E.g., the language of arithmetic
  - \( x+2 \geq y \) is a sentence; \( x^2+y > \varnothing \) is not a sentence.
  - \( x+2 \geq y \) is true iff the number \( x+2 \) is no less than the number \( y \).
  - \( x+2 \geq y \) is true in a world where \( x = 7, y = 1 \).
  - \( x+2 \geq y \) is false in a world where \( x = 0, y = 6 \).
Entailment

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

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

  - E.g., the KB containing “the Giants won” and “the Reds won” entails “Either the Giants won or the Reds won” (\( A, B \) entails \( A \mid B \))
  - E.g., \( x+y = 4 \) entails \( 4 = x+y \)
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics
Models

• Logicians typically think in terms of **models**, which are formally structured worlds with respect to which truth can be evaluated (you can think of “models” as a set of variable assignments).

• 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. $KB = \text{Giants won and Reds won}$
  $\alpha = \text{Giants won}$
Entailment in the wumpus world

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

Consider possible models for $KB$ assuming only pits

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

[Diagram of Wumpus models]
Wumpus models

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

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

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

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

- $KB \models_i \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ by procedure } i$
- **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 $KB \models_i \alpha$
- Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.
- That is, the procedure will answer any question whose answer follows from what is known by the $KB$. 
Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas

- The proposition symbols $P_1, P_2$ etc are sentences
  
  - If $S$ is a sentence, $\neg S$ is a sentence (negation)
  - If $S_1$ and $S_2$ are sentences, $S_1 \land S_2$ is a sentence (conjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \lor S_2$ is a sentence (disjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)
Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

<table>
<thead>
<tr>
<th>$P_{1,2}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

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

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

- $\neg S$ is true iff $S$ is false
- $S_1 \land S_2$ is true iff $S_1$ is true and $S_2$ is true
- $S_1 \lor S_2$ is true iff $S_1$ is true or $S_2$ is true
- $S_1 \Rightarrow S_2$ is true iff $S_1$ is false or $S_2$ is true
  
  i.e., $S_1 \Rightarrow S_2$ 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.,

$$\neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) = true \land (true \lor false) = true \land true = true$$
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>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
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"

\[ B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \]
\[ B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \]
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>true</td>
</tr>
<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>true</td>
</tr>
<tr>
<td>false</td>
<td>true</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>false</td>
<td>true</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>false</td>
<td>true</td>
<td>false</td>
<td>true</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>true</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>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

Enumerate rows (different assignments to symbols). If $KB$ is true in a row, check that $\alpha$ is true also. If so, then $KB \Rightarrow \alpha$
Inference by enumeration

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

```plaintext
function TT-ENTAILS?(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic
    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 $\beta \models \alpha$

```
(\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 \iff \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
```
Validity and satisfiability

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

Validity is connected to inference via the **Deduction Theorem**:
$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid (so this gives a meeting to $KB \models \alpha$, it must be true in all models (assignments to variables))

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:
$KB \nvdash \alpha$ 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
  - **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
    - examples: Conjunctive 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
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):

\[
\begin{array}{c}
\ell_1 \lor \ldots \lor \ell_k, \\
m_1 \lor \ldots \lor m_n
\end{array}
\]

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

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

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

\[
P_{1,3}
\]

- Resolution is sound and complete for propositional logic
Reasoning (& notation)

Modus Ponens:
\[
\alpha \implies \beta, \alpha \\
\hline
\beta
\]

And Elimination:
\[
\alpha \land \beta \\
\hline
\alpha
\]

Biconditional:
\[
\alpha \iff \beta \\
\hline
(\alpha \implies \beta) \land (\beta \implies \alpha)
\]
Resolution

Soundness of resolution inference rule:

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

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

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

where again, \( l_i \) and \( m_j \) are complementary literals
CNF

- conjunctive normal form (CNF)
  - every sentence in prop. logic can be explained in this way.
  - expressed as a conjunction of disjunctions of literals.
  - Ex:
    - $(A \lor B \lor C) \land (\neg A \lor C \lor D \lor E) \land (C \lor E \lor \neg F) \land \ldots$
    - 3-CNF: when each disjunctive clause has only 3 literals.
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 \((\land\) 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., to show that $KB \Rightarrow \alpha$, we show $\neg (KB \Rightarrow \alpha)$, or show $KB \land \neg \alpha$ unsatisfiable. First we represent it as CNF.

```plaintext
function PL-RESOLUTION(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

- \( KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \)
- \( \alpha = \neg P_{1,2} \)
- Top row shows CNF form of KB

- Note that many resolution steps are vacuous (since they resolve to true).
Forward and backward chaining

• **Horn Form** (restricted)
  
  KB = conjunction of Horn clauses
  
  – Horn clause =
    
    • proposition symbol; or
    
    • (conjunction of symbols) ⇒ symbol
    
    • disjunction of literals at which at most one is positive.
    
    • Can be written as an implication where:
      
      – premise is conjunction of positive literals
      
      – conclusion is a single positive literal.
    
  – E.g., \((L_{1,1} \land \text{Breeze}) \Rightarrow B_{1,1}\)

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

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

• Integrity constraint: \((\neg W_{1,1} \lor \neg W_{1,2}) \Rightarrow W_{1,1} \land W_{1,2} \Rightarrow \text{false}\)

• Can be used with **forward chaining** or **backward chaining**.

• These algorithms are very natural and run in **linear time**
Forward chaining

• Goal: trying to prove a premise: Say p.
• Idea: fire any rule whose premises are satisfied in the $KB$,
  – add its

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

```
function PL-FC-ENTAILS?(KB, q) returns true or false
    inputs: KB, the knowledge base, a set of propositional Horn clauses
            q, the query, a proposition symbol
    local variables: count, a table, indexed by clause, initially the number of premises
                     inferred, a table, indexed by symbol, each entry initially false
                     agenda, a list of symbols, initially the symbols known in KB

    while agenda is not empty do
        p ← POP(agenda)
        unless inferred[p] do
            inferred[p] ← true
            for each Horn clause c in whose premise p appears do
                decrement count[c]
                if count[c] = 0 then do
                    if HEAD[c] = q then return true
                    PUSH(HEAD[c], agenda)
            end for
        end unless
        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 all 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 goal query \( q \):

to prove \( q \) by BC:

check if \( q \) is known already, or
prove by BC all premises of some rule that concludes \( q \) 
recurse.

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
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
Efficient propositional inference

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 (think of CNF form for this).
   A clause is true if any literal within clause is true (so don’t need to prove all)
   A sentence is false if any clause is false (don’t need to disprove all)

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 (since if a sentence has a model (making it true), then it has one with the pure symbols assigned to be true).

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true (otherwise clause will be false).
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 \texttt{WalkSAT} algorithm

function \texttt{WalkSAT}(\texttt{clauses, }p, \texttt{max-flips}) \texttt{returns} a satisfying model or \texttt{failure}

\texttt{inputs: clauses, a set of clauses in propositional logic}
\texttt{p, the probability of choosing to do a “random walk” move}
\texttt{max-flips, number of flips allowed before giving up}

\texttt{model} \leftarrow \texttt{a random assignment of true/false to the symbols in clauses}

\texttt{for } i = 1 \texttt{ to } \texttt{max-flips} \texttt{ do}
\texttt{\hspace{1em}if model satisfies clauses then return model}
\texttt{\hspace{1em}clause} \leftarrow \texttt{a randomly selected clause from clauses that is false in model}
\texttt{\hspace{1em}with probability } p \texttt{ flip the value in model of a randomly selected symbol}
\texttt{\hspace{1.5em}from clause}
\texttt{\hspace{1em}else flip whichever symbol in clause maximizes the number of satisfied clauses}
\texttt{return failure}
Hard satisfiability problems

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

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

\(m = \) number of clauses
\(n = \) 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 distinct proposition symbols, 155 sentences
function PL-WUMPUS-AGENT( percept) returns an action

inputs: percept, a list, [stench, breeze, glitter]
static: KB, initially containing the “physics” of the wumpus world
x, y, orientation, the agent’s position (init. [1,1]) and orient. (init. right)
visited, an array indicating which squares have been visited, initially false
action, the agent’s most recent action, initially null
plan, an action sequence, initially empty

update x, y, orientation, visited based on action
if stench then TELL(KB, S_{x,y}) else TELL(KB, \neg S_{x,y})
if breeze then TELL(KB, B_{x,y}) else TELL(KB, \neg B_{x,y})
if glitter then action ← grab
else if plan is nonempty then action ← POP(plan)
else if for some fringe square [i,j], \text{ASK}(KB, (\neg P_{i,j} \land \neg W_{i,j})) is true or
for some fringe square [i,j], \text{ASK}(KB, (P_{i,j} \lor W_{i,j})) is false then do
plan ← A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j], visited))
action ← POP(plan)
else action ← a randomly chosen move

return action
Expressiveness limitation of propositional logic

• KB contains "physics" sentences for every single square

• For every time $t$ and every location $[x,y]$, $L_{x,y}^t \land FacingRight^t \land Forward^t \Rightarrow L_{x+1,y}^t$

• Rapid proliferation of clauses
• This will be solved in first-order logic (next chapter).
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.

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

• Propositional logic lacks expressive power