Logical Agents

Chapter 7
Logical Agents

• What are we talking about, “logical?”
  – Aren’t search-based chess programs logical
    • Yes, but knowledge is used in a very specific way
      – Win the game
      – Not useful for extracting strategies or understanding other aspects of chess
  – We want to develop more general-purpose knowledge systems that support a variety of logical analyses
Why study knowledge-based agents

• Partially observable environments
  – combine available information (percepts) with general knowledge to select actions

• Natural Language
  – Language is too complex and ambiguous. Problem-solving agents are impeded by high branching factor.

• Flexibility
  – Knowledge can be reused for novel tasks. New knowledge can be added to improve future performance.
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

- 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
7.2 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

![Wumpus world diagram]
Exploring a wumpus world
7.3 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; \( x2+y > \{\} \) is not a sentence
  - \( x+2 \geq y \) is true iff the number \( x+2 \) is no less than the number \( y \)
  - \( x+2 \geq y \) is true in a world where \( x = 7, y = 1 \)
  - \( x+2 \geq y \) is false in a world where \( x = 0, y = 6 \)
Entailment

• 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”
  – 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.

- We say $m$ is a model of a sentence $\alpha$.

- $M(\alpha)$ 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
Wumpus models

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

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

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

- $KB = wumpus$-world rules + observations
- $\alpha_2 = "[2,2] is safe"]$, $KB \vdash \alpha_2$
Inference

• $KB \models_i \alpha$ = sentence $\alpha$ can be derived from $KB$ 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 $KB$. 
What is a logic?

• A formal language
  – Syntax – what expressions are legal
  – Semantics – what legal expressions mean
  – Proof system – a way of manipulating syntactic expressions to get other syntactic expressions (which will tell us something new)

• Why proofs? Two kinds of inferences an agent might want to make:
  – Multiple percepts ) conclusions about the world
  – Current state & operator ) properties of next state
Models
Propositional (Boolean) Logic

• Syntax of allowable sentences
  – atomic sentences
    • indivisible syntactic elements
    • Use uppercase letters to represent a proposition that can be true or false
    • True and False are predefined propositions where True means always true and False means always false
Atomic sentences

• Syntax of atomic sentences
  – indivisible syntactic elements
  – Use uppercase letters to represent a proposition that can be true or false
  – True and False are predefined propositions where True means always true and False means always false
Complex sentences

• Formed from atomic sentences using connectives
  – ~ (or = not): the negation
  – ^ (and): the conjunction
  – V (or): the disjunction
  – => (or = implies): the implication
  – ⇔ (if and only if): the biconditional
Backus-Naur Form (BNF)

\[
\begin{align*}
\text{Sentence} & \rightarrow \text{AtomicSentence} \mid \text{ComplexSentence} \\
\text{AtomicSentence} & \rightarrow \text{True} \mid \text{False} \mid \text{Symbol} \\
\text{Symbol} & \rightarrow P \mid Q \mid R \mid \ldots \\
\text{ComplexSentence} & \rightarrow \neg \text{Sentence} \\
& \quad \mid ( \text{Sentence} \land \text{Sentence} ) \\
& \quad \mid ( \text{Sentence} \lor \text{Sentence} ) \\
& \quad \mid ( \text{Sentence} \Rightarrow \text{Sentence} ) \\
& \quad \mid ( \text{Sentence} \Leftrightarrow \text{Sentence} )
\end{align*}
\]
Propositional (Boolean) Logic

• Semantics
  – given a particular model (situation), what are the rules that determine the truth of a sentence?
  – use a truth table to compute the value of any sentence with respect to a model by recursive evaluation
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
  – E.g. \( P_{1,2} \), \( P_{2,2} \), \( P_{3,1} \)
  – false true false
• With these symbols, 8 possible models, can be enumerated automatically.
• Rules for evaluating truth with respect to a model \( m \):
  \[ \neg S \text{ is true iff } S \text{ is false} \]
  \[ S_1 \land S_2 \text{ is true iff } S_1 \text{ is true and } S_2 \text{ is true} \]
  \[ S_1 \lor S_2 \text{ is true iff } S_1 \text{ is true or } S_2 \text{ is true} \]
  \[ S_1 \Rightarrow S_2 \text{ is true iff } S_1 \text{ is false or } S_2 \text{ is true} \]
  i.e., is false iff \( S_1 \text{ is true and } S_2 \text{ is false} \)
  \[ S_1 \Leftrightarrow S_2 \text{ is true iff } S_1 \Rightarrow S_2 \text{ is true and } S_2 \Rightarrow S_1 \text{ is true} \]
• 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>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</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>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>false</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>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</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>false</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
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 \( \alpha \vdash \beta \) and \( \beta \vdash \alpha \)

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

- 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
Propositional inference: normal forms

Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms.

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

\[ \text{conjunction of disjunctions of literals} \]

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

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

\[ \text{disjunction of conjunctions of literals} \]

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

**Horn Form (restricted)**

\[ \text{conjunction of Horn clauses (clauses with } \leq 1 \text{ positive literal)} \]

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

Often written as set of implications:

\[ B \implies A \text{ and } (C \land D) \implies B \]
7.5 Reasoning Pattern in propositional Logic

Proof methods divide into (roughly) two kinds:

Model checking
- truth table enumeration (sound and complete for propositional)
- heuristic search in model space (sound but incomplete)
  e.g., the GSAT algorithm (Ex. 6.15)

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.
7.5 Reasoning Pattern in propositional Logic

• Inference Rules
  – **Modus Ponens:**
    
    Whenever sentences of form $\alpha \Rightarrow \beta$ and $\alpha$ are given
    the sentence $\beta$ can be inferred
    
    – $R_1$: Green $\Rightarrow$ Martian
    – $R_2$: Green
    – Inferred: Martian
Reasoning w/ propositional logic

• Inference Rules
  – And-Elimination
    • Any of conjuncts can be inferred
      – $R_1$: Martian ^ Green
      – Inferred: Martian
      – Inferred: Green

• Use truth tables if you want to confirm inference rules
Example of a proof

• There is no pit in $[1,1]$:
  \[ R_1 : \neg P_{1,1}. \]

• A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:
  \[ R_2 : B_{1,1} \iff (P_{1,2} \lor P_{2,1}). \]
  \[ R_3 : B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1}). \]

• The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in Figure 7.3(b).
  \[ R_4 : \neg B_{1,1}. \]
  \[ R_5 : B_{2,1}. \]
Example of a proof

that is, there is no pit in [1,2]. First, we apply biconditional elimination to $R_2$ to obtain

$$R_6 : \quad (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}).$$

Then we apply And-Elimination to $R_6$ to obtain

$$R_7 : \quad ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}).$$

Logical equivalence for contrapositives gives

$$R_8 : \quad (\neg B_{1,1} \Rightarrow \neg (P_{1,2} \lor P_{2,1})).$$

Now we can apply Modus Ponens with $R_8$ and the percep $R_4$ (i.e., $\neg B_{1,1}$), to obtain

$$R_9 : \quad \neg (P_{1,2} \lor P_{2,1}).$$

Finally, we apply de Morgan’s rule, giving the conclusion

$$R_{10} : \quad \neg P_{1,2} \land \neg P_{2,1}.$$

That is, neither [1,2] nor [2,1] contains a pit.
Inference Rules

◊ **Modus Ponens or Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

\[
\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}
\]

◊ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

\[
\frac{\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n}{\alpha_i}
\]

◊ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

\[
\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n}
\]

◊ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

\[
\frac{\alpha_i}{\alpha_1 \lor \alpha_2 \lor \ldots \lor \alpha_n}
\]
Inference Rules

◊ **Double-Negation Elimination:** (From a doubly negated sentence, you can infer a positive sentence.)

\[
\frac{\neg\neg\alpha}{\alpha}
\]

◊ **Unit Resolution:** (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

\[
\frac{\alpha \lor \beta, \quad \neg\beta}{\alpha}
\]

◊ **Resolution:** (This is the most difficult. Because $\beta$ cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

\[
\frac{\alpha \lor \beta, \quad \neg\beta \lor \gamma}{\alpha \lor \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}
\]
Constructing a proof

• Proving is like searching
  – Find sequence of logical inference rules that lead to desired result
  – Note the explosion of propositions
    • Good proof methods ignore the countless irrelevant propositions

• The fact that inference in propositional logic is NP-complete.

• In many practical cases, finding a proof can be highly efficient simply because it can ignore irrelevant propositions, no matter how many of them.
Monotonicity of knowledge base

• Knowledge base can only get larger
  – Adding new sentences to knowledge base can only make it get larger
    – If (KB entails $\alpha$)
    – $((KB \land \beta) \text{ entails } \alpha)$

• This is important when constructing proofs
  – A logical conclusion drawn at one point cannot be invalidated by a subsequent entailment
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):**

  \[
  \begin{array}{c}
  l_i \lor \ldots \lor l_k, \\
  m_j \lor \ldots \lor m_n
  \end{array}
  \]

  \[
  \frac{\begin{array}{c}
  l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k \lor m_j \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n
  \end{array}}{l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k \lor m_j \lor \ldots \lor m_n}
  \]

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

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

  \[
  \frac{P_{1,3}}{}
  \]

- **Resolution is sound and complete**
  for propositional logic
Resolution

Soundness of resolution inference rule:

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

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

$$\neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n)$$
Conversion to CNF

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

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

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

```
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

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \lnot B_{1,1}$

  $\alpha = \lnot P_{1,2}$
Forward and backward chaining

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

- 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
  – has already been proved true, or
  – 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
7.6 Efficient propositional inference

Two families of efficient algorithms for propositional inference based on model checking:

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

• Hillclimbing search
The DPLL algorithm

- Checking satisfiability:
  - Determine if an input propositional logic sentence (in CNF) is satisfiable.
- DPLL (Davis-Putnam Algorithm)
  - Search through possible assignments to (G, L, M) via depth-first search

- Improvements over truth table enumeration:
  - **Early termination:** avoids examination of entire subtrees in the search space
    - A clause is true if any literal is true. \((A \lor B) \land (A \lor C)\) is true if A is true
    - A sentence is false if any clause is false.
  - **Pure symbol heuristic**
    - **Pure symbol:** always appears with the same "sign" in all clauses.
    - e.g., In the three clauses \((A \lor \neg B), (\neg B \lor \neg C), (C \lor A)\), **A and B are pure**, C is impure.
    - Make a **pure symbol literal true**.
  - **Unit clause heuristic:** assigns all unit clause symbols before branching
    - **Unit clause:** only one literal in the clause, e.g. if B=false, \((B \lor \neg C)\) become a unit clause \((\neg C) \equiv true, C \equiv false\).
    - The only literal in a unit clause must be true.
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
Searching for variable values

• Other ways to find (G, L, M) assignments for:
  \[ G \land L \land (\neg L \lor \neg G \lor M) \land \neg M = 0 \]

  – Simulated Annealing *(WalkSAT)*
  
  • Start with initial guess \((0, 1, 1)\)
  
  • With each iteration, pick an unsatisfied clause and flip one symbol in the clause
  
  • Evaluation metric is the number of clauses that evaluate to true
  
  • Move “in direction” of guesses that cause more clauses to be true
  
  • Many local mins, use lots of randomness
WalkSAT termination

• How do you know when simulated annealing is done?
  – No way to know with certainty that an answer is not possible
    • Could have been bad luck
    • Could be there really is no answer
    • Establish a max number of iterations and go with best answer to that point
The \texttt{WalkSAT} algorithm

\begin{verbatim}
function \texttt{WalkSAT}(\texttt{clauses}, \texttt{p}, \texttt{max-flips}) returns a satisfying model or \texttt{failure}
   inputs: \texttt{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} ← a random assignment of \texttt{true}/\texttt{false} to the symbols in \texttt{clauses}
   for \texttt{i} = 1 to \texttt{max-flips} do
      if \texttt{model} satisfies \texttt{clauses} then return \texttt{model}
         \texttt{clause} ← a randomly selected clause from \texttt{clauses} that is false in \texttt{model}
         with probability \texttt{p} flip the value in \texttt{model} of a randomly selected symbol
         from \texttt{clause}
      else flip whichever symbol in \texttt{clause} maximizes the number of satisfied clauses
   return \texttt{failure}
\end{verbatim}
Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,
  \[(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)\]

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

- 16 of 32 possible assignments are models (are satisfiable) for this sentence
- Therefore, 2 random guesses should find a solution
  - WalkSAT and DPLL should work quickly
Critical point

• Increase $m$ (number of clauses)
  – Hard problems seem to cluster near $m/n = 4.3$ (critical point)
Hard satisfiability problems

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

A wumpus-world agent using propositional logic:

\[ \neg P_{1,1} \]
\[ \neg W_{1,1} \]
\[ B_{x,y} \iff (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y}) \]
\[ S_{x,y} \iff (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y}) \]
\[ W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4} \text{ (at least one wumpus)} \]
\[ \neg W_{1,1} \lor \neg W_{1,2} \text{ (at most one wumpus) } n(n-1)/2 \]
\[ \neg W_{1,1} \lor \neg W_{1,3} \]
\[ \ldots \]

\[ \Rightarrow 64 \text{ distinct proposition symbols, 155 sentences} \]
Terminology

⇒ Fringe squares
⇒ Provably safe
⇒ Possible safe square
Algorithm

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 \leftarrow \text{grab}
else if plan is nonempty then action \leftarrow \text{POP(plan)}
else if for some fringe square [i,j], ASK(KB, (\neg P_{i,j} \land \neg W_{i,j})) is true or
for some fringe square [i,j], ASK(KB, (P_{i,j} \lor W_{i,j})) is false then do
plan \leftarrow A^{*}-GRAPH-SEARCH(Route-PB([x,y], orientation, [i,j], visited))
action \leftarrow \text{POP(plan)}
else action \leftarrow \text{a randomly chosen move}
return action
Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square
- The larger the environment the larger the knowledge base.

Goal: two sentences: for all squares

- For every time \( t \) and every location \([x, y]\),
  \[ L^t_{x,y} \land \text{FacingRight}^t \land \text{Forward}^t \Rightarrow L^t_{x+1,y} \]

- 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 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