Logical Agents

"Artificial Intelligence: A Modern Approach", 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

$$L_A^0 \vee L_B^0 \vee D_A^0 \vee D_B^0$$

Knowledge-based agents

- Reasoning operates on internal representation of knowledge
  - Logic as a general class of representation
    - Propositional logic
    - First-order logic

Knowledge bases

- Knowledge base (KB) = 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

The procedural approach encodes desired behaviors directly as program code

Knowledge bases

- 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
- Inference
  - derives new sentences from old ones
  - Basis of TELL and ASK operations

We dignify a sentence with the name axiom, when the sentence is taken as given without being derived from other sentences

A simple knowledge-based agent

```
function KB-Agent(percept) returns an action
    persistent: 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, percepts,….
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

A simple knowledge-based agent

- MAKE-PERCEPT-SENTENCE
  - Constructs a sentence asserting that the agent perceived the given percept at the given time
- MAKE-ACTION-QUERY
  - Constructs a sentence that asks what action should be done at the current time
- MAKE-ACTION-SENTENCE
  - constructs a sentence asserting that the chosen action was executed

Wumpus world

- Wumpus
- Pitts
- Gold
- Agent

Wumpus world PEAS description

- Performance measure
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow
- Environment
  - 4×4 grid
  - Agent starts in [1,1] while facing to the right
  - Gold and wumpus are located randomly in the squares except to [1,1]
  - Each square other than [1,1] can be a pit with probability 0.2
- Sensors
  - Stench, Breeze, Glitter, Bump, Scream
    - E.g. [Stench, Breeze, None, None, None]
- Actuators
  - Forward, TurnLeft, TurnRight, Shoot, Grab, Climb

Wumpus world characterization

- Observable?

Wumpus world characterization

- Observable? No - only local perception
- Deterministic?

Wumpus world characterization

- Observable? No - only local perception
- Deterministic? Yes - outcomes exactly specified
- Episodic?

Wumpus world characterization

- Observable? No - only local perception
- Deterministic? Yes - outcomes exactly specified
- Episodic? No - sequential at the level of actions
- Static?

Wumpus world characterization

- 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?

Wumpus world characterization

- 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?

Wumpus world characterization

- 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

A: agent
OK: safe square

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze
P: Pit

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze
P: Pit
S: Stench

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze
P: Pit
S: Stench

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze
P: Pit
S: Stench

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze
P: Pit
S: Stench

Exploring a Wumpus world

A: agent
OK: safe square
B: Breeze
P: Pit
S: Stench

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 if the number $x + 2$ is no less than the number y
  - $x + 2 \geq y$ is true in a world where $x = 7; y = 1$
  - $x + 2 \geq y$ is false in a world where $x = 0; y = 6$

Propositional logic: Syntax

- The proposition symbols P, Q, … are sentences
- If P is a sentence, ¬P is a sentence (negation)
- If P and Q are sentences, P ∧ Q is a sentence (conjunction)
- If P and Q are sentences, P ∨ Q is a sentence (disjunction)
- If P and Q are sentences, P ⇒ Q is a sentence (implication)
- If P and Q are sentences, P ⇔ Q is a sentence (biconditional)

Propositional logic (Grammar)

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence & \rightarrow & (\ Sentence\ ) \mid [\ Sentence\ ] \\
 & \mid & \neg\ Sentence \\
 & \mid & Sentence \wedge Sentence \\
 & \mid & Sentence \vee Sentence \\
 & \mid & Sentence \Rightarrow Sentence \\
 & \mid & Sentence \Leftrightarrow Sentence
\end{array}
$$

$$
\text{OPERATOR PRECEDENCE} \quad : \quad \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
$$

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Propositional logic: Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- Each model specifies true/false of each proposition symbol
  - If the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is:
    - m1 = {$P_{1,2}$ = false, $P_{2,2}$ = false, $P_{3,1}$ = true}
  - There are 2^3 = 8 possible models
- Semantics of a complex sentence in any model m:
  - $\neg P$ is true iff P is false in m.
  - $P \wedge Q$ is true iff both P and Q are true in m.
  - $P \vee Q$ is true iff either P or Q is true in m.
  - $P \Rightarrow Q$ is true unless P is true and Q is false in m.
  - $P \Leftrightarrow Q$ is true iff P and Q are both true or both false in m.

Entailment

- Entailment means that one thing follows from another:
  - $\mathrm{KB} \models \alpha$
- Knowledge base KB entails sentence α if and only if α 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

- Models are mathematical abstraction of possible worlds
  - Each model fixes the truth or falsehood of every relevant sentence.
- 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
  - m $\in$ M(α) if α is true in model m
  - E.g, α: x + y = 4, all possible assignments of values to x, y are models. M(α) contains a subset of models satisfying x + y = 4.
- Then KB $\models \alpha$ if and only if M(KB) $\subseteq$ M(α)
  - E.g. KB = Giants won and Reds won
  - α = Giants won

Entailment in the Wumpus world

- Situation after detecting nothing in [1,1], moving right, breeze in [2,1]
- Consider possible models for ?s assuming only pits
- 3 Boolean choices ⇒ 8 possible models

Wumpus models

Wumpus models

KB = wumpus-world rules + observations

Wumpus models

KB = wumpus-world rules + observations

α_1= "[1,2] is safe",

KB $\models$ α_1, proved by model checking

Wumpus models

KB = wumpus-world rules + observations

α_2 = "[2,2] is safe",

KB $\not\models \alpha_2$

Inference

- KB $\vdash_i \alpha$ = sentence α can be derived from KB by procedure i
- Consequences of KB are a haystack; α 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 $\vDash \alpha$
- Completeness: i is complete if
  - Whenever KB $\vDash \alpha$, it is also true that KB $\vdash_i \alpha$

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"
  - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- "A square is breezy if and only if there is an adjacent pit"

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

$KB \models \neg P_{1,2}$?

$KB \models P_{2,2}$?

Inference by enumeration(model checking)

- Depth-first enumeration of all models is sound and complete
- For n symbols, time complexity is $O(2^n)$, space complexity is $O(n)$.

```
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))
```

Logical equivalence

- Two sentences are logically equivalent iff true in same models:
  - $\alpha \equiv \beta$ if and only if $\alpha \vDash \beta$ and $\beta \vDash \alpha$

$(\alpha \wedge \beta)$	$\equiv$	$(\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv$	$(\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv$	$(\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv$	$(\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

Validity and satisfiability

- A sentence is valid if it is true in all models,
  - e.g., True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
- 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., $A \vee B$, C
- A sentence is unsatisfiable if it is true in no models
  - e.g., $A \wedge \neg A$
- Satisfiability is connected to inference via the following:
  - $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

Inference methods

- Inference methods can be categorized into:
  - Model checking (enumerating models)
    - 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
  - Theorem proving (searching proofs by applying inference rules)
    - Applying a sequence of inference rules on KB to find the desired sentence
      - Can use inference rules as operators in a standard search alg
    - Legitimate (sound) generation of new sentences from old ones
    - Typically require translation of sentences into a normal form

Inference rules

- All logical entailments can be used as inference rules.
- Biconditional Elimination

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \qquad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

- Modus Ponens

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta} \qquad \frac{(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot, (WumpusAhead \wedge WumpusAlive)}{Shoot}$$

- And Elimination

$$\frac{\alpha \wedge \beta}{\alpha} \qquad \frac{(WumpusAhead \wedge WumpusAlive)}{WumpusAlive}$$

Example

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

- Can we infer from R_1 through R_5 to prove $\neg P_{1,2}$?

  R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [biconditional elim. to R_2]

Example

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

- Can we infer from R_1 through R_5 to prove $\neg P_{1,2}$?

R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [biconditional elim. to R_2]

R_7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [and elim. to R_6]

Example

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

- Can we infer from R_1 through R_5 to prove $\neg P_{1,2}$?

R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [biconditional elim. to R_2]

R_7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [and elim. to R_6]

R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$ [$(P \Rightarrow Q) \equiv (\neg Q \Rightarrow \neg P)$]

Example

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

- Can we infer from R_1 through R_5 to prove $\neg P_{1,2}$?

R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [biconditional elim. to R_2]

R_7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [and elim. to R_6]

R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$ [$(P \Rightarrow Q) \equiv (\neg Q \Rightarrow \neg P)$]

R_9: $\neg(P_{1,2} \vee P_{2,1})$ [Modus Ponens with R_8 and R_4]

Example

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

- Can we infer from R_1 through R_5 to prove $\neg P_{1,2}$?

R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [biconditional elim. to R_2]

R_7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ [and elim. to R_6]

R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$ [$(P \Rightarrow Q) \equiv (\neg Q \Rightarrow \neg P)$]

R_9: $\neg(P_{1,2} \vee P_{2,1})$ [Modus Ponens with R_8 and R_4]

R_{10}: $\neg P_{1,2} \wedge \neg P_{2,1}$ [De Morgan rule]

Finding a proof as a search problem

- Initial state
  - The initial knowledge base
- Actions
  - All inference rules applied to all the sentences that match their top line
- Result
  - Add the sentence in the bottom line of the used inference rule
- Goal
  - A state containing the sentence we are trying to prove.

Monotonicity property

- The set of entailed sentences can only grow as information is added to KB.
  - Inference rules can be applied where premises are found in KB (regardless of what else is in KB)
- if $KB \models \alpha$ then $KB \wedge \beta \models \alpha$ (for any α and β)
  - The assertion β can not invalidate any conclusion α already inferred.

Resolution

- Inference rules are sound.
- Is a set of rules complete?
  - Are the available inference rules adequate?
- Resolution
  - A single inference rule that yields a complete inference algorithm (when couples with any complete search algorithm).

Resolution

- Unit resolution
  - l_i and m are complementary literals ($l_i = \neg m$)

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

$$\frac{\neg p \vee q \vee r \vee s \quad , \neg r}{\neg p \vee q \vee s}$$

- Full resolution rule:
  - l_i and m_j are complementary literals

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

$$\frac{\neg p \vee q \vee r \vee s \quad , t \vee q \vee \neg r}{\neg p \vee q \vee s \vee t}$$

Resolution-Example

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Resolution

- It resolution rule sound? Why?

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

$l_i = False$ $\qquad\qquad$ $l_i = True$

$\downarrow$ $\qquad\qquad$ $\downarrow$

$m_j = False$

$\downarrow$

$l_1 \vee l_2 \ldots l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \quad \vee \quad m_1 \vee m_2 \ldots m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$

Resolution

- A resolution-based theorem prover can, for any sentences α and β in propositional logic, decide whether $\alpha \models \beta$.
- The next two subsections explain how resolution accomplishes this.
  - CNF (Conjunctive Normal Form)
  - Is $(\alpha \wedge \neg\beta)$ unsatisfiable?

CNF Grammar

- A sentence expressed as a conjunction of clauses is said to be in **conjunctive normal form** or **CNF**

$$\begin{aligned}
\text{CNFSentence} &\rightarrow \text{Clause}_1 \wedge \ldots \wedge \text{Clause}_n \\
\text{Clause} &\rightarrow \text{Literal}_1 \vee \ldots \vee \text{Literal}_m \\
\text{Literal} &\rightarrow \text{Symbol} \mid \neg\text{Symbol} \\
\text{Symbol} &\rightarrow P \mid Q \mid R \mid \ldots
\end{aligned}$$

- E.g.
  - $(P_1 \vee P_2) \wedge (\neg P_3 \vee P_4 \vee P_5) \wedge P_6$

Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $(\neg\alpha \vee \beta)$
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $(\neg\alpha \vee \beta)$
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Move $\neg$ inwards using de Morgan's rules and double-negation
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $(\neg\alpha \vee \beta)$
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Move $\neg$ inwards using de Morgan's rules and double-negation
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Apply distributivity law ($\vee$ over $\wedge$) and flatten
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution algorithm

- Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ is unsatisfiable

```
function PL-RESOLUTION(KB,α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each pair of clauses Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci,Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Resolution algorithm

- Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ is unsatisfiable

function PL-RESOLUTION(KB,α) **returns** true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

Each pair containing complementary literals is resolved and the resulted clause is added to the set if it is not already present.

The process continues until one of these happens:

- No new clauses ($KB \not\models \alpha$)
- Two clauses resolve to yield the empty clause ($KB \models \alpha$)

α

    new $\leftarrow$ new $\cup$ resolvents
    if new $\subseteq$ clauses **then return** false
    clauses $\leftarrow$ clauses $\cup$ new

Resolution example

- KB = $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$
- Convert KB $\wedge \neg\alpha$ into CNF
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) \wedge (\neg B_{1,1}) \wedge P_{1,2}$

Horn clauses & definite clauses

- Some real-world KBs satisfy restrictions on the form of sentences
  - We can use more restricted and efficient inference algorithms
- Definite clause
  - A disjunction of literals of which exactly one is positive literal

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee P_{k+1} \Leftrightarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow P_{k+1}$$

- Horn clause
  - A disjunction of literals of which at most one is positive literal.
  - Closed under resolution

Grammar (HNF)

$$\text{Symbol} \rightarrow P \mid Q \mid R \mid \ldots$$

$$\text{HornClauseForm} \rightarrow \text{DefiniteClauseForm} \mid \text{GoalClauseForm}$$

$$\text{DefiniteClauseForm} \rightarrow (\text{Symbol}_1 \wedge \cdots \wedge \text{Symbol}_l) \Rightarrow \text{Symbol}$$

$$\text{GoalClauseForm} \rightarrow (\text{Symbol}_1 \wedge \cdots \wedge \text{Symbol}_l) \Rightarrow \text{False}$$

Forward and backward chaining

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

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Can be used with forward chaining or backward chaining.
- These algorithms are very natural and run in time that is linear in the size of KB
- They are bases of logic programming
  - Prolog: backward chaining

Forward chaining

- Idea:
  - Fire any rule whose premises are satisfied in the KB
  - Add its conclusion to the KB, until query is found

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
        inputs: KB, the knowledge base, a set of propositional definite clauses
                q, the query, a proposition symbol
        count ← a table, where count[c] is the number of symbols in c's premise
        inferred ← a table, where inferred[s] is initially false for all symbols
        agenda ←a queue of symbols, initially symbols known to be true in KB

        while agenda is not empty do
                p ← POP(agenda)
                if p = q then return true
                if inferred[p] = false then
                        inferred[p] ←true
                        for each clause c in KB where p is in c.PREMISE do
                                decrement count[c]
                                if count[c] = 0 then add c.CONCLUSION to agenda
        return false
```

Begin with known facts (positive literals) in KB

The process continues until q is added or no further inference

Forward chaining example

agenda = {A, B}

inferred = [false, false, false, false, false, false]

A	B	L	M	P	Q

$P \Rightarrow Q$	$count(P \Rightarrow Q) = 1$
$L \wedge M \Rightarrow P$	$count(L \wedge M \Rightarrow P) = 2$
$B \wedge L \Rightarrow M$	$count(B \wedge L \Rightarrow M) = 2$
$A \wedge P \Rightarrow L$	$count(A \wedge P \Rightarrow L) = 2$
$A \wedge B \Rightarrow L$	$count(A \wedge B \Rightarrow L) = 2$
A	A
B	B

Forward chaining example

agenda = {B}

inferred = [true, false, false, false, false, false]

A	B	L	M	P	Q

$P \Rightarrow Q$	count($P \Rightarrow Q$) = 1
$L \wedge M \Rightarrow P$	count($L \wedge M \Rightarrow P$) = 2
$B \wedge L \Rightarrow M$	count($B \wedge L \Rightarrow M$) = 2
$A \wedge P \Rightarrow L$	count($A \wedge P \Rightarrow L$) = 1
$A \wedge B \Rightarrow L$	count($A \wedge B \Rightarrow L$) = 1
A	A
B	B

Forward chaining example

agenda = {L}

inferred = [true, true, false, false, false, false]

A B L M P Q

$P \Rightarrow Q$	count($P \Rightarrow Q$) = 1
$L \wedge M \Rightarrow P$	count($L \wedge M \Rightarrow P$) = 2
$B \wedge L \Rightarrow M$	count($B \wedge L \Rightarrow M$) = 1
$A \wedge P \Rightarrow L$	count($A \wedge P \Rightarrow L$) = 1
$A \wedge B \Rightarrow L$	count($A \wedge B \Rightarrow L$) = 0
A	A
B	B

Forward chaining example

agenda = {M}

inferred = [true, true, true, false, false, false]

A B L M P Q

$P \Rightarrow Q$	$count(P \Rightarrow Q) = 1$
$L \wedge M \Rightarrow P$	$count(L \wedge M \Rightarrow P) = 1$
$B \wedge L \Rightarrow M$	$count(B \wedge L \Rightarrow M) = 0$
$A \wedge P \Rightarrow L$	$count(A \wedge P \Rightarrow L) = 1$
$A \wedge B \Rightarrow L$	$count(A \wedge B \Rightarrow L) = 0$
A	A
B	B

Forward chaining example

agenda = {P}

inferred = [true, true, true, true, false, false]

A	B	L	M	P	Q

$P \Rightarrow Q$	$count(P \Rightarrow Q) = 1$
$L \wedge M \Rightarrow P$	$count(L \wedge M \Rightarrow P) = 0$
$B \wedge L \Rightarrow M$	$count(B \wedge L \Rightarrow M) = 0$
$A \wedge P \Rightarrow L$	$count(A \wedge P \Rightarrow L) = 1$
$A \wedge B \Rightarrow L$	$count(A \wedge B \Rightarrow L) = 0$
A	A
B	B

Forward chaining example

agenda = {L, Q}

inferred = [true, true, true, true, true, false]

A	B	L	M	P	Q

$P \Rightarrow Q$	$count(P \Rightarrow Q) = 0$
$L \wedge M \Rightarrow P$	$count(L \wedge M \Rightarrow P) = 0$
$B \wedge L \Rightarrow M$	$count(B \wedge L \Rightarrow M) = 0$
$A \wedge P \Rightarrow L$	$count(A \wedge P \Rightarrow L) = 0$
$A \wedge B \Rightarrow L$	$count(A \wedge B \Rightarrow L) = 0$
A	A
B	B

Forward chaining example

agenda = {Q}

inferred = [true, true, true, true, true, false]

A	B	L	M	P	Q

$P \Rightarrow Q$	count($P \Rightarrow Q$) = 0
$L \wedge M \Rightarrow P$	count($L \wedge M \Rightarrow P$) = 0
$B \wedge L \Rightarrow M$	count($B \wedge L \Rightarrow M$) = 0
$A \wedge P \Rightarrow L$	count($A \wedge P \Rightarrow L$) = 0
$A \wedge B \Rightarrow L$	count($A \wedge B \Rightarrow L$) = 0
A	A
B	B

Forward chaining example

agenda = {Q}

inferred = [true, true, true, true, true, false]

A	B	L	M	P	Q

$P \Rightarrow Q$	count($P \Rightarrow Q$) = 0
$L \wedge M \Rightarrow P$	count($L \wedge M \Rightarrow P$) = 0
$B \wedge L \Rightarrow M$	count($B \wedge L \Rightarrow M$) = 0
$A \wedge P \Rightarrow L$	count($A \wedge P \Rightarrow L$) = 0
$A \wedge B \Rightarrow L$	count($A \wedge B \Rightarrow L$) = 0
A	A
B	B

Forward chaining

- Assumption:
  - KB of definite clauses
- Sound? yes
  - Every inference is an application of Modus Ponens.
- Complete? yes
  - Every entailed atomic sentence will be derived?

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
     - Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m
       Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m
       Therefore the algorithm has not reached a fixed point!
  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

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward vs. backward chaining

- Forward chaining is data-driven, automatic, unconscious processing
  - May do lots of work that is irrelevant to the goal
- Backward chaining is goal-driven, appropriate for problem-solving
- 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
- They test entailment KB $\models \alpha$ by testing unsatisfiability of KB $\wedge \neg\alpha$

The DPLL algorithm

- Determine if an input propositional logic sentence (in CNF) is satisfiable
- Improvements over truth table enumeration (simple DFS):
  - Early termination
    - A clause is true if any literal 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 $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.
    - Make a pure symbol literal true
  - Unit clause heuristic
    - Unit clause: only one literal in the clause
    - 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, model ∪ {P=value})
   P, value ← FIND-UNIT-CLAUSE(clauses, model)
   if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
   P ← FIRST(symbols); rest ← REST(symbols)
   return DPLL(clauses, rest, model ∪ {P=true}) or
          DPLL(clauses, rest, model ∪ {P=false}))
```

The DPLL algorithm: Example

- Consider the following knowledge base:

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

The DPLL algorithm: Example

$\neg Q$ A B $\neg P \vee Q$ $\neg L \vee \neg M \vee P$ $\neg B \vee \neg L \vee M$ $\neg A \vee \neg P \vee L$ $\neg A \vee \neg B \vee L$

The DPLL algorithm: Example

$\neg Q$ $\quad A \quad B \quad \neg P \vee Q \quad \neg L \vee \neg M \vee P \quad \neg B \vee \neg L \vee M$

$\neg A \vee \neg P \vee L \rightarrow \neg P \vee L$

$\neg A \vee \neg B \vee L \rightarrow \neg B \vee L$

$A = T$

The DPLL algorithm: Example

	$\neg Q$	A	B	$\neg P \vee Q$	$\neg L \vee \neg M \vee P$	$\neg B \vee \neg L \vee M$	$\neg A \vee \neg P \vee L$	$\neg A \vee \neg B \vee L$
$A=T$							$\downarrow$	$\downarrow$
							$\neg P \vee L$	$\neg B \vee L$
						$\downarrow$		$\downarrow$
$B=T$						$\neg L \vee M$		L

The DPLL algorithm: Example

	$\neg Q$	A	B	$\neg P \vee Q$	$\neg L \vee \neg M \vee P$	$\neg B \vee \neg L \vee M$	$\neg A \vee \neg P \vee L$	$\neg A \vee \neg B \vee L$
$A=T$							$\neg P \vee L$	$\neg B \vee L$
$B=T$						$\neg L \vee M$		L
$L=T$					$\neg M \vee P$	M	True	

The DPLL algorithm: Example

	$\neg Q$	A	B	$\neg P \vee Q$	$\neg L \vee \neg M \vee P$	$\neg B \vee \neg L \vee M$	$\neg A \vee \neg P \vee L$	$\neg A \vee \neg B \vee L$
$A=T$							$\neg P \vee L$	$\neg B \vee L$
$B=T$						$\neg L \vee M$		L
$L=T$					$\neg M \vee P$	M	True	
$M=T$					P			

The DPLL algorithm: Example

	$\neg Q$	A	B	$\neg P \vee Q$	$\neg L \vee \neg M \vee P$	$\neg B \vee \neg L \vee M$	$\neg A \vee \neg P \vee L$	$\neg A \vee \neg B \vee L$
$A=T$							$\neg P \vee L$	$\neg B \vee L$
$B=T$						$\neg L \vee M$		L
$L=T$					$\neg M \vee P$	M	True	
$M=T$					P			
$P=T$				Q				

The DPLL algorithm: Example

	$\neg Q$	A	B	$\neg P \vee Q$	$\neg L \vee \neg M \vee P$	$\neg B \vee \neg L \vee M$	$\neg A \vee \neg P \vee L$	$\neg A \vee \neg B \vee L$
$A=T$							$\neg P \vee L$	$\neg B \vee L$
$B=T$						$\neg L \vee M$		L
$L=T$					$\neg M \vee P$	M	True	
$M=T$					P			
$P=T$				Q				
$Q=T$	False							

The WalkSAT algorithm

- Incomplete, local search algorithm
  - state space: complete assignments
- Evaluation function:
  - The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness
  - To escape from local minima

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, typically around 0.5
           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
```

The WalkSAT algorithm

- Failure may mean:
  - The sentence is unsatisfiable.
  - The algorithm needs more time.
- It can not detect unsatisfiability
  - Cannot be reliably used for deciding entailment
  - If max_flips = ∞ and p > 0, WalkSAT will find a model (if any exists)
  - If max_flips = ∞ it never terminates for unsatisfiable sentences

Hard satisfiability problems

- Consider random 3-CNF sentences.
  - *e.g.*, $(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E)$
    $\wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$
- m = number of clauses
- n = number of symbols
- Hard problems seem to cluster near m/n = 4.3 (critical point)

Hard satisfiability problems

Inference-based agents in the Wumpus world

- A Wumpus-world agent using propositional logic:

$$\neg P_{1,1}$$
$$\neg W_{1,1}$$
$$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$$
$$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$$
$$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$$
$$\neg W_{1,1} \vee \neg W_{1,2}$$
$$\neg W_{1,1} \vee \neg W_{1,3}$$
$$\ldots$$

- 64 distinct proposition symbols, 155 sentences

Limitations of propositional logic

- Wumpus world
  - Distinct rules "for each square [x,y]" and "for each time t"
- Many rules are needed to describe the world axioms
  - Rather impractical for bigger world
- We can not express general knowledge about "physics" of the world directly in propositional language
  - More expressive language is needed
- Qualification problem: specifying all the exceptions
  - Probability theory allows us to summarize all the exceptions (without explicitly naming them)