Propositional Logic, First Order Logic, Proof Methods, and Knowledge-based Agents

Prof. M. Walker
How can we represent human knowledge about the world and how to do things?
The Problem with Natural Language

- Ambiguous
  - “I know more intelligent people than him.”
  - “I saw the Grand Canyon flying to New York.”
  - ”I saw a 747 flying to New York.”

=> We need a **formal way of representing the knowledge usually represented in language**
Two schools of thought in AI for the last 50 years

- Symbolic: represent knowledge using logic and rules that define inference procedures on logic
- Statistical: represent knowledge as probabilities based on observations of the world
- Both of these traditions had their roots in the same paper: McCulloch and Pitts 1943
McCulloch Pitts Neuron

Does weighted sum of inputs pass threshold?

Total input = weight on line 1 x input on 1 + weight on line 2 x input on 2 + weight on line n x input on n (for all n)

Basic model: performs weighted sum of inputs, compares this to internal threshold level, and turns on if this level exceeded.
Propositional Logic Calculus

- The symbols of propositional logic are:
  - P, Q, R, S
  - These can stand for any statement. E.g. “it is raining”, “I am at work”

- The truth symbols: T true, F false

- The connectives:
  - AND \( \land \)
  - OR \( \lor \)
  - NOT \( \neg \)
  - IMPLIES \( \Rightarrow \)
  - EQUALS =
Propositional Logic Syntax of Sentences

- True, False, P, Q, R are sentences
- The negation of a sentence is a sentence
  - NOT P (“it is not raining”)
- The conjunction of two sentences is a sentence,
  - P AND Q (“it is raining and I am at work”)
  - P, Q are called the conjuncts
- The disjunction of two sentences is a sentence,
  - P OR Q (“it is raining or I am at work”)
  - P, Q are called the disjuncts
Propositional Logic Syntax of Sentences

- The implication of one sentence from another
  - P IMPLIES Q ("it is sunny implies I am outdoors")
  - P is called the premise/antecedent
  - Q is called the conclusion/consequent

- The equivalence of two sentences is a sentence
  - P OR Q EQUALS R

- Legal sentences are also called well-formed formulae
((P AND Q) IMPLIES R) EQUALS NOT P OR NOT Q OR R

- Is well-formed:

- P, Q, R are propositions
- P AND Q is a sentence
- (P AND Q) IMPLIES R is a sentence
- NOT P, NOT Q are sentences
- NOT P OR NOT Q is a sentence
- NOT P OR NOT Q OR R is a sentence
- ((P AND Q) IMPLIES R) EQUALS NOT P OR NOT Q OR R
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
Pros and cons of propositional logic

😊 Propositional logic is declarative

😊 Propositional logic allows partial/disjunctive/negated information
  - (unlike most data structures and databases)

😊 Propositional logic is compositional:
  - meaning of $B_{1,1} \land P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

😊 Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)

😉 Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "pits cause breezes in adjacent squares"
    - except by writing one sentence for each square
McCulloch Pitts Neuron for AND

\[
X + Y - 2 \rightarrow X \text{ AND } Y
\]
Logical Agents
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. (Tell (KB, Make-percept ....)
- Incorporate new percepts
- Update internal representations of the world (Tell (KB, Make-action..)
- 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 arrow

- **Environment**
  - Squares adjacent to wumpus are smelly (**STENCH**)
  - 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:** Five Item Ordered Vector
  - **Stench, Breeze:** if Wumpus or Pit Adjacent
  - **Glitter:** if Gold in square
  - **Bump:** When agent hits a wall
  - **Scream:** when Wumpus dies perceived everywhere

- **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 (randomly generated)

PERCEPT = [ none, none, none, none, none]
Exploring a wumpus world
Exploring a wumpus world

PERCEPT = [ none, Breeze, none, none, none]
Exploring a wumpus world

Go back to start state where it's safe.
Exploring a wumpus world

• Go to square 1,2, you know it's okay
• PERCEPT = [Stench, none, none, none, none, none]
• Because there is no breeze, Agent knows that square 2,2 does not have a pit
• Because there was no stench in square 1,2, agent knows there is no W in square 2,2
Exploring a wumpus world

• Go to square 2,2, you know its okay
• PERCEPT = [ none, none, none, none, none]
• Because there is no breeze Agent knows that square 2,3 and 3,1 does not have a pit
Exploring a wumpus world

• Infer that 2,3 and 3,2 are also okay
Exploring a wumpus world
Entailment

- **Entailment** means that one thing **follows from** another:

\[ \text{KB} \models \alpha \]

- Knowledge base \( \text{KB} \) entails sentence \( \alpha \) if and only if \( \alpha \) is true in all worlds where \( \text{KB} \) 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* $\alpha$ if $\alpha$ is true in *m*.

- $M(\alpha)$ is the set of all models of $\alpha$.

- Then $\text{KB} \models \alpha$ iff $M(\text{KB}) \subseteq M(\alpha)$.

  - E.g. $\text{KB} = \text{Giants won and Reds won}$
  - $\alpha = \text{Giants won}$
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: what is possibly true given percepts

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

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

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

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

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

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 $KB$. 
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"

\[
B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \\
B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})
\]
Inference by enumeration

- Depth-first enumeration of all models is **sound and complete**
- **Soundness:** *if the inference procedure says it is entailed, it is*
- **Completeness:** *if it is entailed, the inference procedure can prove it*

```
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 \(O(2^n)\) (have to check all possible assignments)
  - 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**:
  \( KB \vdash \alpha \) 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:
  \( KB \vdash \alpha \) 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

- 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):

\[
\frac{\ell_i \lor \ldots \lor \ell_k, \quad m_1 \lor \ldots \lor m_n}{\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n}
\]

where \(\ell_i\) and \(m_j\) are complementary literals.
- E.g., \(P_{1,3} \lor P_{2,2}, \neg P_{2,2}\)

- Resolution is sound and complete for propositional logic
Conversion to CNF;FIX

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

1. Eliminate \( \iff \), replacing \( \alpha \iff \beta \) with \( (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \).

\((B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})\)

2. Eliminate \( \Rightarrow \), replacing \( \alpha \Rightarrow \beta \) with \( \neg \alpha \lor \beta \).

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

3. Move \( \neg \) inwards using de Morgan's rules and double-negation:

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

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

```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
    end loop
```
Resolution example

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1}))) \land \neg B_{1,1} \quad \alpha = \neg P_{1,2}$
- $\alpha$ = what we want to prove, no pit in 1,2,, ie. $\neg P_{1,2}$
Forward and backward chaining

- **Horn Form (restricted)**
  - $\text{KB} = \text{conjunction of Horn clauses}$
  - 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**

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

\[
\beta
\]

- Can be used with **forward chaining or backward chaining**.
- These algorithms are very natural and run in **linear time**
Forward chaining

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

\[
P \rightarrow Q \\
L \land M \rightarrow P \\
B \land L \rightarrow M \\
A \land P \rightarrow L \\
A \land B \rightarrow L \\
A \\
B
\]
Forward chaining algorithm

function PL-FC-ENTAILS?(KB, q) returns true or false
    local variables: count, a table, indexed by clause, initially the number of premises
    inferred, a table, indexed by symbol, each entry initially false
    agenda, a list of symbols, initially the symbols known to be true
    
    while agenda is not empty do
        p ← POP(agenda)
        unless inferred[p] do
            inferred[p] ← true
            for each Horn clause c in whose premise p appears do
                decrement count[c]
                if count[c] = 0 then do
                    if HEAD[c] = q then return true
                    PUSH(HEAD[c], agenda)
        
    return false

- Forward chaining is sound and complete for Horn KB
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Proof of completeness

- FC derives every atomic sentence that is entailed by $KB$

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model $m$, assigning true/false to symbols
3. Every clause in the original $KB$ is true in $m$
   $$a_1 \land \ldots \land a_k \Rightarrow b$$
4. Hence $m$ is a model of $KB$
5. If $KB \models q$, $q$ is true in every model of $KB$, including $m$
Backward chaining

Idea: work backwards from the query $q$:

- to prove $q$ by BC,
  - check if $q$ is known already, or
  - prove by BC all premises of some rule concluding $q$

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
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

EVERY FOL KB can be converted to a PL KB

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

Improvements over truth table enumeration:

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

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

3. **Unit clause heuristic**
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.
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 \textbf{WalkSAT} algorithm

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

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

return failure
\end{verbatim}
Today’s Thought Question

- What kind of knowledge cannot easily be represented in propositional logic?
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.