A brief history of reasoning

450 B.C.  Stoics  propositional logic, inference (maybe)
322 B.C.  Aristotle  “syllogisms” (inference rules), quantifiers
1565  Cardano  probability theory (propositional logic + uncertainty)
1847  Boole  propositional logic (again)
1879  Frege  first-order logic
1922  Wittgenstein  proof by truth tables
1930  Gödel  $\exists$ complete algorithm for FOL
1930  Herbrand  complete algorithm for FOL (reduce to propositional)
1931  Gödel  $\neg\exists$ complete algorithm for arithmetic
1960  Davis/Putnam  “practical” algorithm for propositional logic
1965  Robinson  “practical” algorithm for FOL—resolution
Conjunctive Normal Form (CNF—universal)

conjunction of \underline{disjunctions} of literals

clauses

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

Resolution inference rule (for CNF): complete for propositional logic

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

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

where \(\ell_i\) and \(m_j\) are complementary literals. E.g.,

\[
P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2}
\]

\[
P_{1,3}
\]

Resolution is sound and complete for propositional logic
Conversion to CNF

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

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

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

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

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

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

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

4. Apply distributivity law (\( \lor \) over \( \land \)) 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
  inputs: KB, the knowledge base, a sentence in propositional logic
          \alpha, the query, a sentence in propositional logic

  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

\[ KB = (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \land \neg B_{1,1} \land \neg P_{1,2} \]
Efficient propositional inference

Two families of efficient algorithms for propositional inference:

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

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

Improvements over truth table enumeration:

1. Early termination
   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 **WalkSAT** algorithm

```plaintext
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
```
Hard satisfiability problems

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

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

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

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

- Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$
Pros and cons of propositional logic

Propositional logic is **declarative**: pieces of syntax correspond to facts

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
First-order logic

Whereas propositional logic assumes world contains facts, first-order logic (like natural language) assumes the world contains

- **Objects**: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
- **Relations**: red, round, bogus, prime, multistoried . . ., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, . . .
- **Functions**: father of, best friend, third inning of, one more than, end of . . .
Syntax of FOL: Basic elements

Constants  \textit{KingJohn, 2, UCB,}…
Predicates  \textit{Brother, >,}…
Functions  \textit{Sqrt, LeftLegOf,}…
Variables  \(x, y, a, b,\)…
Connectives  \(\land, \lor, \neg, \Rightarrow, \Leftrightarrow\)
Equality  \(=\)
Quantifiers  \(\forall, \exists\)
Deducing hidden properties

Properties of locations:
\[ \forall x, t \ (At(Agent, x, t) \land Smelt(t)) \Rightarrow Smelly(x) \]
\[ \forall x, t \ (At(Agent, x, t) \land Breeze(t)) \Rightarrow Breezy(x) \]

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect
\[ \forall y \ Breezy(y) \Rightarrow \exists x \ Pit(x) \land \text{Adjacent}(x, y) \]

Causal rule—infer effect from cause
\[ \forall x, y \ Pit(x) \land \text{Adjacent}(x, y) \Rightarrow Breezy(y) \]

Neither of these is complete—e.g., the causal rule doesn’t say whether squares far away from pits can be breezy

Definition for the Breezy predicate:
\[ \forall y \ Breezy(y) \iff [\exists x \ Pit(x) \land \text{Adjacent}(x, y)] \]
Knowledge base for the wumpus world

“Perception”
\[ \forall b, g, t \ (\text{Percept}([\text{Smell}, b, g], t) \Rightarrow \text{Smelt}(t)) \]
\[ \forall s, b, t \ (\text{Percept}([s, b, \text{Glitter}], t) \Rightarrow \text{AtGold}(t)) \]

Reflex: \[ \forall t \ \text{AtGold}(t) \Rightarrow \text{Action(Grab}, t) \]

Reflex with internal state: do we have the gold already?
\[ \forall t \ \text{AtGold}(t) \land \neg \text{Holding(Gold}, t) \Rightarrow \text{Action(Grab}, t) \]

\text{Holding(Gold, t)} cannot be observed
\[ \Rightarrow \text{keeping track of change is essential} \]
Reduction to propositional inference

Suppose the KB contains just the following:

\[
\forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \\
King(John) \\
Greedy(John) \\
Brother(Richard, John)
\]

Instantiating the universal sentence in all possible ways, we have

\[
King(John) \land Greedy(John) \Rightarrow Evil(John) \\
King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \\
King(John) \\
Greedy(John) \\
Brother(Richard, John)
\]

The new KB is propositionalized: proposition symbols are

\[
King(John), Greedy(John), Evil(John), King(Richard) \text{ etc.}
\]
Claim: a ground sentence* is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

Idea: propositionalize KB and query, apply resolution, return result

Problem: with function symbols, there are infinitely many ground terms, e.g., \( \text{Father(Father(Father(John)))} \)

Theorem: Herbrand (1930). If a sentence \( \alpha \) is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB

Idea: For \( n = 0 \) to \( \infty \) do
- create a propositional KB by instantiating with depth-\( n \) terms
- see if \( \alpha \) is entailed by this KB

Problem: works if \( \alpha \) is entailed, loops if \( \alpha \) is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable
Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences.
E.g., from

\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]
\[ King(John) \]
\[ \forall y \ Greedy(y) \]
\[ Brother(Richard, John) \]

it seems obvious that \( Evil(John) \), but propositionalization produces lots of facts such as \( Greedy(Richard) \) that are irrelevant

With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations

With function symbols, it gets much much worse!
Resolution: brief summary

Full first-order version:

\[
\frac{\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n}{(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta}
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

For example,

\[
\begin{align*}
-\text{Rich}(x) \lor \text{Unhappy}(x) \\
\text{Rich}(Ken) \\
\hline
\text{Rich}(Ken) \\
\text{Unhappy}(Ken)
\end{align*}
\]

with \( \theta = \{x/Ken\} \)

Apply resolution steps to \( \text{CNF}(KB \land \neg \alpha) \); complete for FOL
Conversion to CNF

Everyone who loves all animals is loved by someone:

\[ \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg \forall y \ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]

2. Move \( \neg \) inwards:

\( \neg \forall, p \equiv \exists x \ \neg p \), \( \neg \exists, p \equiv \forall x \ \neg p \):

\[ \forall x \ [\exists y \ \neg(\neg Animal(y) \lor Loves(x, y))] \lor [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ \neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]
3. Standardize variables: each quantifier should use a different one

\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists z \ Loves(z, x)] \]

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

\[ \forall x \ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x) \]

5. Drop universal quantifiers:

\[ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x) \]

6. Distribute \(\land\) over \(\lor\):

\[ [Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)] \]