Logical Agents

Chapter 7 (continued)
Outline: Inference

- Resolution in CNF
  - Sound and Complete
- Forward and Backward Chaining using Modus Ponens in Horn Form
  - Sound and Complete
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
Inference by enumeration

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

```python
function TT-ENTAILS?(KB, \alpha) returns true or false
    symbols ← a list of the proposition symbols in KB and \alpha
    return TT-CHECK-ALL(KB, \alpha, symbols, [])

function TT-CHECK-ALL(KB, \alpha, symbols, model) returns true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(\alpha, model)
        else return true
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, \alpha, rest, EXTEND(P, true, model) and
                               TT-CHECK-ALL(KB, \alpha, rest, EXTEND(P, false, model))
```

- For \( n \) symbols, time complexity is \( O(2^n) \), space complexity is \( O(n) \)

This is a Model Checking version of proof
Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$\neg P_{1,1}$
$\neg B_{1,1}$
$B_{2,1}$

"Pits cause breezes in adjacent squares"

$B_{1,1} \iff (P_{1,2} \lor P_{2,1})$
$B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})$
### Truth tables for inference

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$KB$</th>
<th>$\alpha_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</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>true</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>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</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>false</td>
<td>false</td>
</tr>
</tbody>
</table>

$R_1 = \neg P_{1,1}$
$R_4 = \neg B_{1,1}$
$R_5 = B_{2,1}$

$\alpha_1 = P_{1,2}?$
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
Reasoning Patterns in Prop Logic

<table>
<thead>
<tr>
<th>Given(s)</th>
<th>Conclusion</th>
<th>Rules that allow us to introduce new propositions while preserving truth values: logically equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>A \implies B, A</td>
<td>B</td>
<td>Two Examples:</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>- Modus Ponens</td>
</tr>
<tr>
<td>B \land A</td>
<td>A</td>
<td>- And Elimination</td>
</tr>
</tbody>
</table>

Two Examples:
- Modus Ponens
- And Elimination
Logical equivalence

- Two sentences are **logically equivalent** iff true in same models: \( \alpha \equiv \beta \iff \alpha \models \beta \text{ and } \beta \models \alpha \)

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

Conjunctive Normal Form (CNF)
conjunction of disjunctions of literals
clauses
E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

- **Resolution** inference rule (for CNF):

\[
\begin{array}{c}
\ell_i \lor \ldots \lor \ell_k, \\
\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_n
\end{array}
\]

\[
\ell_i \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n
\]

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

\[
P_{1,3}
\]

- Resolution is sound and complete
for propositional logic
Resolution example

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$ (negate the premise for proof by refutation)
The power of false

- Given: \((P) \land (\neg P)\)
- Prove: \(Z\)

Can we prove \(\neg Z\) using the givens above?

| \(\neg P\) | Given       |
| \(P\)     | Given       |
| \(\neg Z\) | Given       |
|            | Unsatisfiable |
Applying inference rules

Equivalent to a search problem

- KB state = node
- Inference rule application = edge

KB:
\[ B, A \land D \land C, \quad B \Rightarrow F \]

KB: \[ B, A \land D \land C, \quad B \Rightarrow F, \quad A \]

KB: \[ B, A \land D \land C, \quad B \Rightarrow F, \quad F \]
Inference

- Define: $KB \vdash_i \alpha = \text{sentence } \alpha \text{ can be derived from } KB \text{ by procedure } i$

- **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 $KB$.

- **Is a set of inference operators complete and sound?**
Completeness

Completeness: \( i \) is complete if whenever \( KB \models \alpha \), it is also true that \( KB \models_i \alpha \)

- An incomplete inference algorithm cannot reach all possible conclusions
  - Equivalent to completeness in search (chapter 3)
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):
\[
\begin{align*}
\ell_1 \lor \ldots \lor \ell_k, & \quad m_1 \lor \ldots \lor m_n \\
\ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n \\
\end{align*}
\]

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

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

Soundness of resolution inference rule:

\[ \neg(\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k) \Rightarrow \ell_i \]

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

\[ \neg(\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

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

- What if \( \ell_i \) and \( \neg m_j \) are false?
- What if \( \ell_i \) and \( \neg m_j \) are true?
Completeness of Resolution

- That is, that resolution can decide the truth value of $S$

- $S = \text{set of clauses}$
- $RC(S) = \text{Resolution closure of } S = \text{Set of all clauses that can be derived from } S \text{ by the resolution inference rule.}$
- $RC(S) \text{ has finite cardinality (finite number of symbols } P_1, P_2, \ldots P_k\text{), thus resolution refutation must terminate.}$
Completeness of Resolution (cont)

- Ground resolution theorem = if $S$ unsatisfiable, $RC(S)$ contains empty clause.
- Prove by proving contrapositive:
  - i.e., if $RC(S)$ doesn’t contain empty clause, $S$ is satisfiable
  - Do this by constructing a model:
    - For each $P_i$, if there is a clause in $RC(S)$ containing $\neg P_i$ and all other literals in the clause are false, assign $P_i = \text{false}$
    - Otherwise $P_i = \text{true}$
  - This assignment of $P_i$ is a model for $S$. 

Forward and backward chaining

- **Horn Form** (restricted)
  - $KB = \text{conjunction of Horn clauses}$
  - 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
  
  \[ a_1, \ldots, a_n, a_1 \land \ldots \land a_n \Rightarrow \beta \]

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

- FC derives every atomic sentence that is entailed by \( KB \) (only for clauses in Horn form)
  1. FC reaches a fixed point (the deductive closure) 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 example
Backward chaining example
Backward chaining example
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
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 \((A \lor \neg B), (\neg B \lor \neg C), (C \lor A)\), \(A\) and \(B\) are pure, \(C\) is impure.
   Make a pure symbol literal true.

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.

What are correspondences between DPLL and in general CSPs?

- Least constraining value
- Most constrained value
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

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

Let’s ask ourselves: Why is it incomplete?
Hard satisfiability problems

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

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

\(m = \) number of clauses
\(n = \) number of symbols

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

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

A wumpus-world agent using propositional logic:

\[-P_{1,1}\]
\[-W_{1,1}\]

\[B_{x,y} \iff (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y})\]

\[S_{x,y} \iff (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y})\]

\[W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4}\]

\[\neg W_{1,1} \lor \neg W_{1,2}\]

\[\neg W_{1,1} \lor \neg W_{1,3}\]

\[\ldots\]

\[\Rightarrow 64 \text{ distinct proposition symbols, 155 sentences}\]
function PL-WUMPUS-AGENT( percept) returns an action

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

update x, y, orientation, visited based on action
if stench then TELL(KB, Sx,y) else TELL(KB, ¬ Sx,y)
if breeze then TELL(KB, Bx,y) else TELL(KB, ¬ Bx,y)
if glitter then action ← grab
else if plan is nonempty then action ← POP(plan)
else if for some fringe square [i,j], ASK(KB, (¬ Pi,j ∧ ¬ Wi,j)) is true or
    for some fringe square [i,j], ASK(KB, (Pi,j ∨ Wi,j)) is false then do
    plan ← A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j], visited))
    action ← POP(plan)
else action ← a randomly chosen move
return action
We didn’t keep track of location and time in the KB. To do this we need more variables:
- $L_{1,1}$ to show that agent in $L_{1,1}$. Does this work?

KB contains "physics" sentences for every single square

For every time $t$ and every location $[x,y]$

$L^t_{x,y} \land FacingRight^t \land Forward^t \Rightarrow L^t_{x+1,y}$

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.