Introduction to Artificial Intelligence

Chapter 3: Knowledge Representation and Reasoning

(2) Propositional Logic (cont)

Nguyễn Hải Minh, Ph.D
nhminh@fit.hcmus.edu.vn
Outline

❑ Horn Clauses
❑ Forward and Backward Chaining
❑ DPLL Algorithm
Horn Clauses

KB:

Disjunctions of literals

$$(l_1 \lor l_2 \lor \cdots \lor l_m)$$

Conjunction Normal Form (CNF)

Clause 1 $\land$ Clause 2 $\land \cdots \land$ Clause $n$

Disjunctions of literals of which at most one is positive

$$(\neg l_1 \lor \neg l_2 \lor \cdots \lor l_m)$$

Horn Clause

E.g., $\neg B_{1,2} \lor \neg B_{2,1} \lor P_{2,2} \iff B_{1,2} \land B_{2,1} \Rightarrow P_{2,2}$
Horn Clauses

- Modus Ponens for Horn Form:

  \[(B \Rightarrow A), B \quad \quad \implies \quad A\]

- More general version of the rule:

  \[(B_1 \land B_2 \land \ldots \land B_k \Rightarrow A), \quad B_1, B_2, \ldots, B_k \quad \implies \quad A\]

- Why do we need Horn Clauses?
  - Horn clauses are closed under resolution
  - In the implication form, the sentence is easier to understand
  - Can be used with forward chaining or backward chaining.
  - These algorithms are very natural and run in linear time.
A grammar for CNF, Horn Clauses

\[
\begin{align*}
CNFSentence & \rightarrow \ Clause_1 \land \cdots \land \ Clause_n \\
Clause & \rightarrow \ Literal_1 \lor \cdots \lor \ Literal_m \\
Literal & \rightarrow \ Symbol \mid \neg Symbol \\
Symbol & \rightarrow \ P \mid Q \mid R \mid \ldots \\
HornClauseForm & \rightarrow \ DefiniteClauseForm \mid \ GoalClauseForm \\
DefiniteClauseForm & \rightarrow \ (Symbol_1 \land \cdots \land Symbol_l) \Rightarrow \ Symbol \\
GoalClauseForm & \rightarrow \ (Symbol_1 \land \cdots \land Symbol_l) \Rightarrow \ False
\end{align*}
\]
Forward chaining

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

query → $P \Rightarrow Q$

$KB \rightarrow$

- $L \land M \Rightarrow P$
- $B \land L \Rightarrow M$
- $A \land P \Rightarrow L$
- $A \land B \Rightarrow L$
- $A$
- $B$
Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
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
```

- Forward chaining is sound and complete for Horn KB
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[
P \Rightarrow Q
\]
\[
L \land M \Rightarrow P
\]
\[
B \land L \Rightarrow M
\]
\[
A \land P \Rightarrow L
\]
\[
A \land B \Rightarrow L
\]
\[
A
\]
\[
B
\]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B &
\end{align*}
\]
Forward chaining example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Forward chaining example

\[
\begin{align*}
    P & \Rightarrow Q \\
    L \land M & \Rightarrow P \\
    B \land L & \Rightarrow M \\
    A \land P & \Rightarrow L \\
    A \land B & \Rightarrow L \\
    A & \\
    B & 
\end{align*}
\]
Forward chaining example

\[
P \Rightarrow Q
\]
\[
L \land M \Rightarrow P
\]
\[
B \land L \Rightarrow M
\]
\[
A \land P \Rightarrow L
\]
\[
A \land B \Rightarrow L
\]
\[
A
\]
\[
B
\]
Backward chaining

**Idea:** work backwards from the query \( q \):

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

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

**Avoid repeated work:** check if new subgoal

- has already been proved true, or
- has already failed
Backward chaining example

\[
P \implies Q \\
L \land M \implies P \\
B \land L \implies M \\
A \land P \implies L \\
A \land B \implies L \\
A \\
B
\]
Backward chaining example

\[
P \implies Q \\
L \land M \implies P \\
B \land L \implies M \\
A \land P \implies L \\
A \land B \implies L \\
A \\
B
\]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Backward chaining example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B & 
\end{align*}
\]
Backward chaining example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Backward chaining example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Forward vs. backward chaining

- **Forward 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

- **Backward chaining:**
  - 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

- The **SAT** problem (checking satisfiability)
  - Testing if $\text{KB} |= \alpha$
  - This can be done by testing unsatisfiability of $\text{KB} \land \neg \alpha$

- Two families of efficient algorithms for propositional inference:
  1. Complete backtracking search algorithms
    - **DPLL** algorithm (*Davis, Putnam, Logemann, Loveland*)
  2. Incomplete local search algorithms (hill-climbing)
    - **WalkSAT** algorithm
The DPLL algorithm

- Often called the *Davis-Putnam algorithm* (1960)
- Determine if an input propositional logic sentence (in CNF) is satisfiable
  - A recursive, depth-first enumeration of possible models.
- Improvements over truth table enumeration:
  1. Early termination
  2. Pure symbol heuristic
  3. Unit clause heuristic
The DPLL algorithm

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

Example:
   - \((A \lor B) \land (A \lor C)\) is true if \(A\) is true, regardless \(B\) and \(C\)

→ Avoid examination of entire subtrees in the search space
The DPLL algorithm

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.

$\rightarrow$ Make a pure symbol literal true

$\rightarrow$ Doing so can never make a clause false
The DPLL algorithm

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

Example:
   - If the model contains \( B = \text{true} \) then \((\neg B \lor \neg C)\) simplifies to \( \neg C \), which is a unit clause.
   - \( C \) must be false (so that \( \neg C = \text{true} \)).
   - Then \( A \) must be true (so that \( C \lor A \) is true).

\( \Rightarrow \) Unit propagation
The DPLL algorithm

```python
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, { })
```

```python
function DPLL(clauses, symbols, model) returns true or false

1. Early Termination

    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false

2. Find-Pure-Symbol

    P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols - P, model ∪ {P=value})

3. Find-Unit-Clause

    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 \texttt{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, 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

- When WalkSAT returns a model
  - The input sentence is Satisfiable
- When it returns false:
  - The sentence is unsatisfiable OR
  - We need to give it more time

→ Most useful when we expect a solution to exist
Inference-based agents in the Wumpus world

A wumpus-world agent using propositional logic:

\[
\begin{align*}
\neg P_{1,1} \\
\neg W_{1,1} \\
B_{x,y} & \iff (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y}) \\
S_{x,y} & \iff (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y}) \\
W_{1,1} & \lor W_{1,2} \lor \ldots \lor W_{4,4} \\
\neg W_{1,1} & \lor \neg W_{1,2} \\
\neg W_{1,1} & \lor \neg W_{1,3} \\
& \ldots
\end{align*}
\]

\[\Rightarrow 64\] distinct proposition symbols, 155 sentences
Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

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

  \[ L_{x,y} \land \text{FacingRight}^t \land \text{Forward}^t \implies L_{x+1,y} \]

- 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 with respect to 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.