Vorlesung
Grundlagen der
Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055

Wintersemester 2012/13 23.11.2012
Chapter 7 (3rd ed., cont‘d)

Logics
From last lecture we know

- Wumpus world
- Propositional Logic with syntax and semantics
- Truth-table approach for proofs
Inference by enumeration

Depth-first enumeration of all models is sound and complete

```plaintext
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 $O(2^n)$, space complexity is $O(n)$
Logical equivalence

- Two sentences are \textit{logically equivalent} if and only if they are true in the same models: \( \alpha \equiv \beta \iff \alpha \models \beta \) and \( \beta \models \alpha \)

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

- Countable alphabet $\Sigma$ of consisting of A, B, C, ...
- Atom: atomic sentence, i.e. a symbol from the alphabet
- Literal: (possibly negated) atomic sentence, e.g. A, $\neg B$
- Clause: Disjunction of literals, e.g. $(A \lor \neg B \lor C)$
Validity and satisfiability

- A sentence is **valid (tautology)** if it is true in all models, e.g., *True, A ∨¬A, A ⇒ A, (A ∧ (A ⇒ B)) ⇒ B*

- A sentence is **satisfiable** if it is true in some model e.g., A ∨ B, C

- A sentence is **unsatisfiable** if it is true in no models e.g., A ∧¬A

- Validity is connected to inference via the Deduction Theorem: 
  \[ KB \models \alpha \text{ if and only if } (KB \Rightarrow \alpha) \text{ is valid} \]

- Satisfiability is connected to inference via: 
  \[ KB \models \alpha \text{ if and only if } (KB \land \neg \alpha) \text{ is unsatisfiable} \]
Proof methods

Proof methods divide into (roughly) two kinds:

- **Application of inference rules**
  - Legitimate (sound) generation of new sentences from existing ones
  - **Proof** = a sequence of inference rule applications
  - Algorithms 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) algorithm
  - heuristic search in model space (sound but incomplete)
    - e.g., min-conflicts-like hill-climbing algorithms
Normal forms

Conjunctive normal form (CNF):
The sentence consists of a conjunction of disjunctions of literals $l_{i,j}$, i.e. it has the following form:

$$\bigwedge_{i=1..n} \left( \bigvee_{j=1..m} l_{i,j} \right)$$

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

Disjunctive normal form (DNF):
The sentence consists of a disjunction of conjunctions of literals $l_{i,j}$, i.e. it has the following form:

$$\bigvee_{i=1..n} \left( \bigwedge_{j=1..m} l_{i,j} \right)$$

e.g., $(A \land B \land \neg C) \lor (\neg D \land B) \lor (E \land C \land F)$
Normal forms - properties

- For all sentences, there is at least one equivalent sentence in CNF and DNF.

- A sentence in DNF is satisfiable iff a disjunction is satisfiable.

- A sentence in KNF is valid iff every conjunction is valid.
Construction of CNF sentences

1. Eliminate $\Rightarrow$ and $\Leftrightarrow$:
   \[ A \Rightarrow B \quad \Rightarrow \quad (\neg A \lor B) \]

2. Move $\neg$ to the inside:
   \[ \neg (A \land B) \quad \Rightarrow \quad (\neg A \lor \neg B) \]

3. Distribute $\lor$ over $\land$:
   \[ (A \land B) \lor C \quad \Rightarrow \quad (A \lor C) \land (B \lor C) \]

4. Simplify:
   \[ (A \lor A) \quad \Rightarrow \quad A \]

The result is a conjunction of disjunction of literals.

- An analogous procedure transforms any sentence into an equivalent sentence in DNF.
- Sentences can be expanded exponentially during the transformation.
Inference rules

Modus ponens
\[ A \implies B, \ A \]
\[ \frac{\ A}{\ B} \]

Conjunction Elimination
\[ A_1 \land A_2 \land A_3 \land \ldots \land A_n \]
\[ \frac{\ A_i}{\ A_1 \lor A_2 \lor A_3 \lor \ldots \lor A_n} \]

Disjunction Introduction
\[ A_i \]
\[ \frac{\ A_1 \lor A_2 \lor A_3 \lor \ldots \lor A_n}{\ A_1 \land A_2 \land A_3 \land \ldots \land A_n} \]
Inference rules - Resolution

Unit resolution

\[
A \lor B, \neg A
\]

\[
\frac{}{B}
\]

Resolution

\[
A \lor B, \neg B \lor C
\]

\[
\frac{}{A \lor C}
\]

or, more general

\[
A_1 \lor A_2 \lor \ldots \lor A_n, \neg A_1 \lor B_2 \lor \ldots \lor B_n
\]

\[
\frac{}{A_2 \lor \ldots \lor A_n \lor B_2 \lor \ldots \lor B_n}
\]
Proof by Resolution - Idea

- We now want to study a deduction technique that does not rely on explicitly testing all interpretations.
- **Idea**: You try to show that a set of sentences is unsatisfiable.
- **However**: It is required that all sentences are given in CNF.
- **But**: In most cases, the sentences are close to their CNF (and there is a quick transformation that preserves satisfiability).
- **Nevertheless**: In the worst case, this deduction technique also requires an exponential amount of time (probably, you cannot avoid this).
Proof by Resolution

To prove that $\text{KB} \models A$, show that $(\text{KB} \land \neg A)$ is unsatisfiable.

Use the inference rules including the resolution rules to infer the empty clause $\Box$ from $(\text{KB} \land \neg A)$.

E.g. show that in the Wumpus world there is no pit at [1,2], formal: $\neg P_{1,2}$.

The Knowledge base is

$$\text{KB} = (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$$
Proof by Resolution

Add the negated clause you want to show to the KB:
A simple resolution algorithm

\begin{function}
\textbf{function} PL-\textsc{Resolution}(KB, \alpha) \textbf{returns} true or false
\begin{align*}
\text{clauses} & \leftarrow \text{the set of clauses in the CNF representation of } KB \wedge \neg \alpha \\
\text{new} & \leftarrow \{ \} \\
\text{loop} & \text{do} \\
& \text{for each } C_i, C_j \text{ in clauses do} \\
& \quad \text{resolvents} \leftarrow \text{PL-\textsc{Resolve}(}C_i, C_j) \\
& \quad \text{if resolvents contains the empty clause then return true} \\
& \quad \text{new} \leftarrow \text{new} \cup \text{resolvents} \\
& \quad \text{if new } \subseteq \text{ clauses then return false} \\
& \quad \text{clauses} \leftarrow \text{clauses} \cup \text{new}
\end{align*}
\end{function}
Completeness of the resolution

- Resolution theorem:

If a set of clauses is unsatisfiable, the resolution closure of these clauses contains the empty clause

- i.e., the resolution is complete, \( KB \models A \) can be shown by deriving \( \square \) from \( (KB \land \neg A) \).
Resticted forms of clauses

- **Definite clause**: disjunction of literals with **exactly** one positive literal, e.g. \((A \lor \neg B \lor \neg C)\)
- **Horn clause**: disjunction of literals with **at most** one positive literal, e.g. \((A \lor \neg B \lor \neg C), \ (\neg B \lor \neg C)\)

The resolvent of two Horn clauses is again a Horn clause.

Horn clauses can be read as implications:
\[A \lor \neg B \lor \neg C \equiv (B \land C) \Rightarrow A\]

Horn clauses can be used for forward- and 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

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
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)
  end for
end unless

return false

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

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.
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 \textit{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
```
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
- DPLL and WalkSAT algorithms