Artificial Intelligence and Machine Learning (CS52)

Term: October 2024 – Jan 2025

Unit 2 - Forward Chaining, Backward Chaining and Resolution

Faculty Coordinator : Jamuna S Murthy

Forward Chaining

Definition: A data-driven inference technique where reasoning starts from known facts and applies inference rules to generate new facts until a goal is reached.

How it works:

1. Start with an initial set of known facts.
2. Match these facts against the conditions (antecedents) of rules.
3. If a rule's conditions are satisfied, the rule 'fires' and its conclusion (consequent) is added to the set of facts.
4. Repeat the process until the desired goal is reached or no new facts can be inferred.

Use Case:

Suitable for scenarios where all the data is available upfront, and the goal is to determine what can be inferred from it.

Example:

Rules:
1. If A and B, then C.
2. If C and D, then E.

Facts: A, B, D.

Inference:
- From A and B, infer C.
- From C and D, infer E.

Backward Chaining

Definition: A goal-driven inference technique where reasoning starts from a goal (a hypothesis) and works backward to determine if the goal can be satisfied using known facts and rules.

How it works:

1. Start with a goal (something you want to prove or achieve).
2. Look for rules that can conclude the goal.
3. For each rule, check if the conditions (antecedents) of the rule are satisfied.
4. If the conditions aren't directly satisfied, treat them as sub-goals and repeat the process recursively.

5. Stop when all conditions are satisfied, proving the goal, or when no further inference is possible.

Use Case:
Ideal for diagnostic and troubleshooting systems where the goal is to verify hypotheses.

Example:
Goal: E.

Rules:
1. If A and B, then C.
2. If C and D, then E.

Facts: A, B, D.

Inference:
- To prove E, need C and D.
- D is a known fact; need to prove C.
- C can be proved from A and B, which are known facts.

Comparison

Aspect	Forward Chaining	Backward Chaining
Direction of Reasoning	Data to Goal	Goal to Data
Driven By	Available facts	Desired goal
Usage	Data exploration, planning	Diagnosis, problem-solving
Efficiency	May generate irrelevant conclusions	Focused on proving specific goals
Examples	Production systems, expert systems	Prolog programming, medical diagnosis

Forward Chaining

1. **Convert all the facts into first-order definite clauses, and then use a forward-chaining algorithm to reach the goal. There are some facts give here:**

1. It is a crime for an American to sell weapons to an enemy of America
2. Country A is the enemy of America
3. A colonel of the army sells missiles to A
4. Missiles are weapons
5. The colonel is an American citizen.

To solve this problem using **forward chaining**:

Step 1: Translate facts into first-order definite clauses

We represent the facts and rules in first-order predicate logic:

1. **Crime Rule**:
   $American(x) \land Weapon(y) \land Enemy(z,America) \land Sells(x,y,z) \Rightarrow Criminal(x)$
2. **Enemy of America**:
   $Enemy(A,America)$
3. **Colonel sells missiles to A**:
   $Sells(Colonel,Missiles,A)$
4. **Missiles are weapons**:
   $Weapon(Missiles)$
5. **The colonel is an American citizen**:
   $American(Colonel)$

Step 2: Forward chaining to reach the goal

We apply forward chaining to infer new facts until we reach the conclusion that the colonel is a criminal.

Initial facts:

- $Enemy(A,America)$
- $Sells(Colonel,Missiles,A)$
- $Weapon(Missiles)$
- $American(Colonel)$

Inferences:

1. **From** Weapon(Missiles): Missiles are identified as weapons.
2. **From** Sells(Colonel,Missiles,A): The colonel sold missiles to Country A.
3. **From** American(Colonel) and Enemy(A,America): Country A is the enemy of America, and the colonel is an American.
4. **Applying the Crime Rule**:
   Since:
   - The colonel is American: American(Colonel),
   - Missiles are weapons: Weapon(Missiles),
   - Country A is the enemy of America: Enemy(A,America),
   - The colonel sold missiles to A: Sells(Colonel,Missiles,A),
5. **Conclusion**: Criminal(Colonel)

Final Answer:

Using forward chaining, we conclude that the colonel is a criminal Criminal(Colonel).

---

2. Consider the following sentences:

1.John likes all kinds of food

2.Apples are food

3.Chicken is food

4.Anything anyone eats and isn't killed by is food.

5.Bill eats peanut and is still alive

6. sue eats everything bill eats

(i) Translate these sentences into formulas in predicate logic.

(ii) Prove that john likes peanuts **using forward chaining**

(i) Translate sentences into predicate logic

We use the following predicates:

- Likes(x,y): x likes y.
- Food(x): x is food.
- Eats(x,y): x eats y.
- KilledBy(x,y): x is killed by y.

Now, translate each sentence:

1. **John likes all kinds of food:**
   $\forall y(Food(y) \Rightarrow Likes(John,y))$
2. **Apples are food:**
   Food(Apples)
3. **Chicken is food:**
   Food(Chicken)
4. **Anything anyone eats and isn't killed by is food:**
   $\forall x \forall y(Eats(x,y) \land \neg KilledBy(x,y) \Rightarrow Food(y))$
5. **Bill eats peanuts and is still alive:**
   $Eats(Bill,Peanuts) \land \neg KilledBy(Bill,Peanuts)$
6. **Sue eats everything Bill eats:**
   $\forall y(Eats(Bill,y) \Rightarrow Eats(Sue,y))$

(ii) Prove that John likes peanuts using forward chaining

Initial facts:

1. Food(Apples)
2. Food(Chicken)
3. $Eats(Bill,Peanuts) \land \neg KilledBy(Bill,Peanuts)$
4. $\forall y(Eats(Bill,y) \Rightarrow Eats(Sue,y))\forall y$
5. $\forall x \forall y(Eats(x,y) \land \neg KilledBy(x,y) \Rightarrow Food(y))$
6. $\forall y(Food(y) \Rightarrow Likes(John,y))$

Steps using forward chaining:

1. **From sentence 5**:
   $Eats(Bill,Peanuts) \land \neg KilledBy(Bill,Peanuts) \Rightarrow Food(Peanuts)$

Since Bill eats peanuts and is still alive, infer: Food(Peanuts).

2. **From sentence 6**:
   $Food(Peanuts) \land \forall y(Food(y) \Rightarrow Likes(John,y))$.

Since John likes all kinds of food and peanuts are food, infer: Likes(John,Peanuts).

Conclusion:

Using forward chaining, we have proven that Likes(John,Peanuts), i.e., John likes peanuts.

Backward Chaining

1. **Convert all the facts into first-order definite clauses, and then use a backward-chaining algorithm to prove that the colonel is committing a crime by selling missiles to Country Al. There are some facts give here:**

1. It is a crime for an American to sell weapons to an enemy of America
2. Country A is the enemy of America
3. A colonel of the army sells missiles to A
4. Missiles are weapons
5. The colonel is an American citizen.

Step 1: Translate facts into first-order definite clauses

We use the following predicates:

- Crime(x): x is a crime.
- American(x): x is an American.
- Sells(x,y,z): x sells yy to z.
- Weapon(y): y is a weapon.
- Enemy(z,w): z is an enemy of w.

Given facts translated:

1. **It is a crime for an American to sell weapons to an enemy of America:**
   $\forall x,y,z(\text{American}(x) \land \text{Weapon}(y) \land \text{Enemy}(z,\text{America}) \land \text{Sells}(x,y,z) \Rightarrow \text{Crime}(\text{Sell}(x,y,z)))$.
2. **Country A is the enemy of America:**
   Enemy(A,America).
3. **A colonel of the army sells missiles to A:**
   Sells(Colonel,Missiles,A).
4. **Missiles are weapons:**
   Weapon(Missiles).
5. **The colonel is an American citizen:**
   American(Colonel).

Step 2: Prove that the colonel is committing a crime using backward chaining

Goal: Prove Crime(Sell(Colonel,Missiles,A)).

Backward-Chaining Steps:

1. **Start with the goal:**
   To prove Crime(Sell(Colonel,Missiles,A)), use the clause:
   American(x)∧Weapon(y)∧Enemy(z,America)∧Sells(x,y,z) ⟹ Crime(Sell(x,y,z)).
   We need to prove the following subgoals:
   - American(Colonel),
   - Weapon(Missiles),
   - Enemy(A,America),
   - Sells(Colonel,Missiles,A).
2. **Subgoal 1: American(Colonel)**
   Given American(Colonel)(fact 5), this is directly true.
3. **Subgoal 2: Weapon(Missiles)**
   Given Weapon(Missiles)(fact 4), this is directly true.
4. **Subgoal 3: Enemy(A,America)**
   Given Enemy(A,America)(fact 2), this is directly true.
5. **Subgoal 4: Sells(Colonel,Missiles,A)**
   Given Sells(Colonel,Missiles,A)(fact 3), this is directly true.
6. **Conclusion:** Since all subgoals are true, we can infer:
   Crime(Sell(Colonel,Missiles,A)).

Final Answer:

Using backward chaining, we have proven that the colonel is committing a crime by selling missiles to Country A.

2. Consider the following sentences:

1.John likes all kinds of food

2.Apples are food

3.Chicken is food

4.Anything anyone eats and isn't killed by is food.

5.Bill eats peanut and is still alive

6. sue eats everything bill eats

(i) Translate these sentences into formulas in predicate logic.

(ii) Prove that john likes peanuts using Backward Chaining.

Proving John Likes Peanuts Using Backward Chaining

(i) Translate Sentences into Predicate Logic

We use the following predicates:
- Likes(x, y): x likes y.
- Food(x): x is food.
- Eats(x, y): x eats y.
- KilledBy(x, y): x is killed by y.

Now, translate each sentence:

1. John likes all kinds of food:
   $\forall y$ (Food(y) $\rightarrow$ Likes(John, y))

2. Apples are food:
   Food(Apples)

3. Chicken is food:
   Food(Chicken)

4. Anything anyone eats and isn't killed by is food:
   $\forall x \ \forall y$ (Eats(x, y) $\wedge \neg$KilledBy(x, y) $\rightarrow$ Food(y))

5. Bill eats peanuts and is still alive:
   Eats(Bill, Peanuts) $\wedge \neg$KilledBy(Bill, Peanuts)

6. Sue eats everything Bill eats:

$\forall y$ (Eats(Bill, y) → Eats(Sue, y))

(ii) Prove That John Likes Peanuts Using Backward Chaining

Goal: Likes(John, Peanuts)

Step 1: Start with the goal
To prove Likes(John, Peanuts), use the formula:
$\forall y$ (Food(y) → Likes(John, y)).
This means we need to prove:
Food(Peanuts).

Step 2: Prove Food(Peanuts)
From the formula $\forall x \forall y$ (Eats(x, y) ∧ ¬KilledBy(x, y) → Food(y)),

we know:
If Eats(Bill, Peanuts) ∧ ¬KilledBy(Bill, Peanuts), then Food(Peanuts).

From the fact Eats(Bill, Peanuts) ∧ ¬KilledBy(Bill, Peanuts) (sentence 5),

we directly infer:
Food(Peanuts).

Step 3: Prove Likes(John, Peanuts)
Since Food(Peanuts) is true, by $\forall y$ (Food(y) → Likes(John, y)),

we conclude:
Likes(John, Peanuts).

Conclusion:

Using backward chaining, we have proven that Likes(John, Peanuts), i.e., John likes peanuts.

Resolution in First Order Logic (Inference Rules)

Resolution in First Order Logic (FOL) is a fundamental inference rule used in automated reasoning. It involves deducing new conclusions by combining existing clauses in a knowledge base. The process extends the concept of propositional resolution to FOL by accounting for variables, quantifiers, and predicates.

Key Concepts in FOL Resolution

1. 1. Clause Normal Form (CNF):

Before applying resolution, sentences in FOL must be converted into CNF, which consists of conjunctions of disjunctions of literals. This involves:

- Eliminating implications and equivalences.
- Reducing the scope of negations.
- Standardizing variables (renaming variables to avoid confusion).
- Moving quantifiers outward (prenex form).
- Skolemization (eliminating existential quantifiers by introducing Skolem constants/functions).
- Distributing disjunctions over conjunctions.

2. Unification:

- To resolve clauses with variables, unification is required.
- A unifier is a substitution that makes two literals identical.
- The most general unifier (MGU) is the simplest unifier that generalizes the substitution.

3. Resolution Rule:

If two clauses contain complementary literals (e.g., P(x) and ¬P(y)), they can be resolved into a new clause by:

- Unifying the literals.
- Removing the complementary pair.
- Combining the remaining literals.

Steps for Resolution in FOL

1. Convert the Knowledge Base (KB) into CNF: Ensure all logical statements are in a suitable form for resolution.
2. Negate the Query: Assume the negation of the query and add it to the KB. This helps in proving the query by contradiction.
3. Apply Resolution: Identify two clauses with complementary literals. Resolve the clauses using unification. Add the resulting clause to the KB if it's not a tautology.

4. Repeat: Continue resolving until either:
   - A contradiction (empty clause) is derived, proving the query.
   - No further resolutions are possible, meaning the query is not entailed by the KB.
5. Conclusion: If an empty clause is derived, the negated query leads to a contradiction, and the original query is true.

Example of FOL Resolution

KB:

$\forall x$ (Human(x) $\Rightarrow$ Mortal(x))

Human(Socrates)

Query:

Mortal(Socrates)?

Resolution:

- Convert KB to CNF:
  ¬Human(x) $\vee$ Mortal(x)
  Human(Socrates)
- Negate the query:
  ¬Mortal(Socrates)
- Combine KB with negated query:
  ¬Human(x) $\vee$ Mortal(x)
  Human(Socrates)
  ¬Mortal(Socrates)
- Resolve:
  ¬Mortal(Socrates) and Mortal(x): ¬Human(Socrates)
  ¬Human(Socrates) and Human(Socrates): Contradiction.
- Conclusion:
  The empty clause is derived, so Mortal(Socrates) is true.

Applications

- Automated theorem proving.
- AI reasoning systems.
- Query answering in logic-based databases.
- Verification of software and hardware systems.

Resolution Refutation

Resolution Refutation is a method of automated theorem proving that uses the resolution principle to derive a contradiction, thereby proving the original statement. It operates by proving the unsatisfiability of a set of clauses, demonstrating that a negated query cannot coexist with the given knowledge base.

Steps in Resolution Refutation

1. Express the Knowledge Base (KB) in Clause Normal Form (CNF): Convert all statements in the KB into CNF. CNF is a conjunction of disjunctions of literals, suitable for resolution.
2. Negate the Query: Assume the negation of the query is true. Add this negation to the KB, turning the problem into proving a contradiction.
3. Resolve Clauses: Apply the resolution rule repeatedly to the KB and the negated query. Look for pairs of clauses with complementary literals (e.g., P(x) and ¬P(y)). Use unification to resolve these clauses, deriving new clauses.
4. Check for the Empty Clause: Continue the resolution process until one of the following happens:
   - The empty clause (a contradiction) is derived, proving the original query.
   - No new clauses can be generated, indicating that the query is not entailed by the KB.
5. Conclude: If the empty clause is derived, the query is proven true because the negated query leads to a contradiction. If the process halts without finding a contradiction, the query is false or cannot be proven.

Example: Resolution Refutation

Knowledge Base (KB):

1. $P(x) \vee Q(x)$

2. $\neg P(a)$

3. $\neg Q(a)$

Query:

Is $P(a) \vee Q(a)$ true?

Steps:

- Negate the Query: Negate $P(a) \vee Q(a)$, resulting in $\neg(P(a) \vee Q(a))$, which simplifies to $\neg P(a) \wedge \neg Q(a)$.
- Add Negation to KB: Add $\neg P(a)$ and $\neg Q(a)$ to the KB (both are already part of the KB).

- Resolve Clauses:
  - From P(x) ∨ Q(x) and ¬P(a), resolve to Q(a).
  - From Q(a) and ¬Q(a), resolve to the empty clause.
- Empty Clause Found: The contradiction proves the query P(a) ∨ Q(a) is true.

Key Features of Resolution Refutation

- Soundness: The method guarantees that if the empty clause is derived, the query logically follows from the KB.
- Completeness: If the query can be derived from the KB, the method will find the empty clause.

Applications of Resolution Refutation

- Automated Theorem Proving: Proves theorems by contradiction.
- Logic Programming: Underlies systems like Prolog for deducing answers.
- Formal Verification: Verifies the correctness of systems by testing logical consistency.
- Artificial Intelligence: Used for reasoning and query answering.

Resolution Refutation Problems :

1. Consider the below argument and using Resolution Refutation Procedure check whether 'John is happy'

"Anyone passing his history exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. John did not study, but John is lucky. Anyone who is lucky wins the lottery."

Step 1: Represent the Problem in Predicate Logic

1. Anyone passing his history exams and winning the lottery is happy:

   ∀x (Pass(x, HistoryExam) ∧ Win(x, Lottery) → Happy(x))

2. Anyone who studies or is lucky can pass all his exams:

   ∀x (Study(x) ∨ Lucky(x) → Pass(x, HistoryExam))

3. John did not study, but John is lucky:

   ¬Study(John), Lucky(John)

4. Anyone who is lucky wins the lottery:

   ∀x (Lucky(x) → Win(x, Lottery))

Query: Is Happy(John)?

Negate the query: ¬Happy(John)

Step 2: Convert to Clause Normal Form (CNF)

1. $\forall x$ (Pass(x, HistoryExam) $\wedge$ Win(x, Lottery) $\rightarrow$ Happy(x)):

   $\neg$Pass(x, HistoryExam) $\vee$ $\neg$Win(x, Lottery) $\vee$ Happy(x)

2. $\forall x$ (Study(x) $\vee$ Lucky(x) $\rightarrow$ Pass(x, HistoryExam)):

   $\neg$Study(x) $\vee$ $\neg$Lucky(x) $\vee$ Pass(x, HistoryExam)

3. $\forall x$ (Lucky(x) $\rightarrow$ Win(x, Lottery)):

   $\neg$Lucky(x) $\vee$ Win(x, Lottery)

Grounded Facts:

$\neg$Study(John)

Lucky(John)

$\neg$Happy(John)

Step 3: Express as Clauses

C1: $\neg$Pass(x, HistoryExam) $\vee$ $\neg$Win(x, Lottery) $\vee$ Happy(x)

C2: $\neg$Study(x) $\vee$ $\neg$Lucky(x) $\vee$ Pass(x, HistoryExam)

C3: $\neg$Lucky(x) $\vee$ Win(x, Lottery)

C4: $\neg$Study(John)

C5: Lucky(John)

C6: $\neg$Happy(John)

Step 4: Perform Resolution

Resolve C2 with C4 and C5:

Substituting x = John:

C2 = $\neg$Study(John) $\vee$ $\neg$Lucky(John) $\vee$ Pass(John, HistoryExam)

C4 = $\neg$Study(John)

C5 = Lucky(John)

Resolving $\neg$Study(John) and Lucky(John):

Pass(John, HistoryExam)

Resolve C3 with C5:

Substituting x = John:

C3 = ¬Lucky(John) ∨ Win(John, Lottery)

C5 = Lucky(John)

Resolving Lucky(John):

Win(John, Lottery)

Resolve C1 with Pass(John, HistoryExam) and Win(John, Lottery):

C1 = ¬Pass(John, HistoryExam) ∨ ¬Win(John, Lottery) ∨ Happy(John)

Pass(John, HistoryExam)

Win(John, Lottery)

Resolving Pass(John, HistoryExam) and Win(John, Lottery):

Happy(John)

Resolve Happy(John) with C6:

C6 = ¬Happy(John)

Happy(John)

This results in an empty clause.

Conclusion

The contradiction proves that John is indeed happy. The Resolution Refutation process has successfully demonstrated this result.

2. All teachers are good. Anyone who is good and intelligent will deliver an excellent lecture. Jamuna is an intelligent teacher. Show that Jamuna will deliver an excellent lecture.

Step 1: Represent the Problem in Predicate Logic

Facts and Rules:

1. **All teachers are good**:
   $\forall x(\text{Teacher}(x) \Rightarrow \text{Good}(x))$
2. **Anyone who is good and intelligent will deliver excellent lectures**:
   $\forall x(\text{Good}(x) \wedge \text{Intelligent}(x) \Rightarrow \text{ExcellentLecture}(x))$
3. **Jamuna is an intelligent teacher**:
   Teacher(Jamuna), Intelligent(Jamuna)

Query:

ExcellentLecture(Jamuna)?

Negate the query:
$\neg$ExcellentLecture(Jamuna)

Step 2: Convert to Clause Normal Form (CNF)

Step-by-Step Conversion:

1. $\forall$**x(Teacher(x)$\Rightarrow$Good(x))**:
   Equivalent to $\neg\text{Teacher}(x) \vee \text{Good}(x)\neg\text{Teacher}(x) \vee \text{Good}(x)$.
2. $\forall$**x(Good(x)$\wedge$Intelligent(x)$\Rightarrow$ExcellentLecture(x))**:
   Equivalent to $\neg\text{Good}(x) \vee \neg\text{Intelligent}(x) \vee \text{ExcellentLecture}(x)$.
3. Grounded Facts:
   - Teacher(Jamuna)
   - Intelligent(Jamuna)
   - Negated Query: $\neg$ExcellentLecture(Jamuna)

Step 3: Express as Clauses

Clauses:

1. C1:$\neg\text{Teacher}(x) \vee \text{Good}(x)$
2. C2:$\neg\text{Good}(x) \vee \neg\text{Intelligent}(x) \vee \text{ExcellentLecture}(x)$
3. C3:Teacher(Jamuna)
4. C4:Intelligent(Jamuna)

5. C5:¬ExcellentLecture(Jamuna)

Step 4: Perform Resolution

Resolve C1 with C3:

Substituting x=Jamuna:

C1=¬Teacher(Jamuna)∨Good(Jamuna)

C3=Teacher(Jamuna)

Resolving Teacher(Jamuna):

Good(Jamuna)

Resolve C2 with Good(Jamuna)Good(Jamuna) and C4:

Substituting x=Jamuna:

C2=¬Good(Jamuna)∨¬Intelligent(Jamuna)∨ExcellentLecture(Jamuna)

Good(Jamuna),Intelligent(Jamuna)

Resolving Good(Jamuna) and Intelligent(Jamuna):

ExcellentLecture(Jamuna)

Resolve ExcellentLecture(Jamuna) with C5:

C5=¬ExcellentLecture(Jamuna)

ExcellentLecture(Jamuna)

This results in an **empty clause**, indicating a contradiction.